Trabajo de Fin de Máster

Titulación

Estudio y desarrollo de los algoritmos de visión y de Inteligencia Artificial aplicados a un robot, para resolver partidas de ajedrez hombre-máquina.

MEMORIA

Autor: Jorge Eduardo Sanango Peña
Director: Rita María Planas Dangla
Convocatoria: abril 2019

Escuela Superior de Ingenierías Industrial, Aeroespacial y Audiovisual de Terrassa

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

Resumen

El presente proyecto consiste en el desarrollo de un aplicativo de algoritmos de visión y de inteligencia artificial aplicados a un robot, para resolver partidas de ajedrez hombre-máquina. Este presenta tres etapas que son: 1) aplicativos de visión artificial para reconocimiento del tablero, algoritmos de reconocimiento de movimientos de piezas e integración de un motor de ajedrez, 2) Creación de una interfaz de usuario, interfaz de comunicación y una unidad central de control, 3) Programación de movimientos del robot y diseño de un elemento terminal. Adicionalmente, se explica como se integran todas estas etapas en el funcionamiento del aplicativo.

En la primera etapa del desarrollo del aplicativo se estudia los algoritmos de visión por computadora, mismos que serán los encargados de reconocer el tablero de ajedrez (líneas, esquinas y casillas), es decir, transformar una imagen en bits y que pueda ser procesada por un ordenador. Después se desarrollará un algoritmo de reconocimiento de movimiento de piezas, que compara dos imágenes, determina los cambios en el color y escribe el movimiento realizado en formato algebraico de ajedrez. Se integrará un motor de ajedrez que interpretará los movimientos realizados por la persona y devolverá una jugada adecuada al movimiento realizado.

La segunda parte consiste en la creación de una interfaz gráfica de usuario, que su utilizará para que las personas puedan interactuar con el back-end del programa, todos los procedimientos que hacen que el programa realice todas las tareas. También se crea una interfaz de comunicación, que servirá para comunicar el aplicativo con la controladora del robot, se utilizará para ello sockets que permiten la comunicación entre dos elementos terminales. Finalmente, se desarrolla una unidad central de control, que servirá como puente entre todos los módulos creados, como son: procesamiento de imágenes, motor de ajedrez, interfaz de usuario y la interfaz de comunicación.

La tercera parte del desarrollo del aplicativo consiste en la programación de movimientos que va a efectuar el robot en el transcurso de una partida, en la metodología de programación de estos movimientos se tiene en cuenta aspectos como dimensiones del robot, tamaño de piezas y tablero de ajedrez, velocidad de operación y tipo de movimiento a realizar. Además, se diseña un elemento terminal para el robot (gripper), que será el responsable de la sujeción de las piezas, para ello se analizará la forma del objeto a sujetar, su peso y se decidirá el cuál es el diseño más adecuado a partir de estos parámetros.

Por último, las tres etapas del diseño del aplicativo se integran de la siguiente manera: la interfaz gráfica de usuario permite la interacción del usuario con el programa, al inicio del juego se analiza el tablero de ajedrez, después de realizado un movimiento se detecta los cambios en la imagen y se envía cuál fue el movimiento realizado, seguido el motor de ajedrez procesa el movimiento y devuelve una jugada, esa jugada es transmitida al robot por medio de la interfaz de comunicación, una vez obtenido el movimiento el robot desplaza la pieza haciendo uso para ello del gripper.

Índice general

Índice de figuras

Índice de tablas

1. Glosario

API:	Interfaz de programación de aplicaciones (Application programming interface).
Back end:	Es la parte trasera de cualquier programa o página web. Se trata de todo el conjunto de desarrollo que se encarga de que un programa haga lo que hace y funcione correctamente.
ELO:	Método matemático, basado en cálculo estadístico, para calcular la habilidad relativa de los jugadores de deportes como el ajedrez.
Enroque:	Es un movimiento especial en el juego de ajedrez que involucra al rey y a una de las torres del jugador. Es el único movimiento en ajedrez en el que el jugador mueve dos piezas a la vez.
Escaque:	Casilla cuadrada de un tablero de ajedrez o damas.
FlexPendant:	El FlexPendant es una unidad de operador de mano que se usa para realizar muchas de las tareas implicadas en el manejo de un sistema de robot: ejecutar programas, mover el manipulador, modificar programas del robot, etc.
Front end:	Es la parte gráfica que se muestra al usuario. Desde aquí el usuario puede interactuar con el programa y ver como este va funcionando.
Gripper:	Es un dispositivo que sostiene un objeto de modo que se pueda manipular. Tiene la capacidad de retener y liberar un objeto mientras se ejecuta alguna acción.
GUI:	Interfaz gráfica de usuario (Graphical User Interface)
IDE:	Entorno de desarrollo integrado (Integrated Development Environment).
POO:	Programación orientada a objetos.
ROI:	Región de interés (Region of Interest).
TCP:	Tool Center Point. Este punto suele definirse como una parte determinada de la herramienta, por ejemplo, la boquilla de una pistola de adhesivo, el centro de una pinza o el extremo de una herramienta rectificadora.
UCI:	Interfaz Universal de Ajedrez (Universal Chess Interface).

2. Introducción

El presente proyecto consiste en el desarrollo de una aplicación de visión artificial que, en conjunto con un motor de ajedrez y un brazo robótico, será capaz de interactuar con una persona en una partida de ajedrez en el mundo real. Para conseguir este objetivo, se utilizará conocimientos de programación, tanto para el reconocimiento de piezas de ajedrez, jugadas y los movimientos que realizará el brazo robótico. Adicionalmente se diseñará una pinza robótica que ayudará una mejor sujeción de las piezas mientras se desarrolla la partida.

2.1. Objetivos del proyecto

- Estudiar los principios de procesado de imágenes, con el fin de procesar las mismas para que sean tratadas por el ordenador.
- Diseñar y desarrollar un módulo de visión artificial que permitirá el reconocimiento del tablero de ajedrez, así como las posiciones en las que se encuentran las piezas.
- Integrar un motor de ajedrez que será el responsable de interpretar los movimientos del jugador humano y que devolverá una jugada adecuada al movimiento recibido.
- Crear un módulo que permita la conexión del motor de ajedrez con la visión artificial.
- Establecer la comunicación entre la controladora del robot y el aplicativo donde se ejecutan los algoritmos de visión artificial en conjunto con el motor de ajedrez.
- Realizar la programación del brazo robótico para que sea capaz de leer la información enviada por el aplicativo y transformarla en movimientos del mismo.
- Implementar una interfaz gráfica de usuario que permita la interacción de la persona con el software de procesado de imágenes, motor de ajedrez y la controladora del robot
- Diseñar un elemento terminal que permita una adecuada sujeción de las piezas de ajedrez, mismo que será instalado en robot.

2.2. Alcance del proyecto

El presente trabajo abarcará varios aspectos para el desarrollo del mismo, de los que se listan los siguientes:

- La programación del proyecto será realizada en Python, un lenguaje de programación ampliamente extendido y utilizado en la actualidad para el desarrollo de aplicaciones de inteligencia artificial.
- Se utilizará OpenCV, una biblioteca de software libre multiplataforma, que en el caso de este proyecto correrá sobre el intérprete de Python, para el procesado de imágenes.
- Se utilizará un tablero de ajedrez de 45x45 [cm] con un escaque de 50 [mm]. Las piezas serán de colores, verdes y rojas, con unas dimensiones de rey de: altura 9,6 [cm], base 3,7 [cm].
- El robot a usarse será un IRB 140 de ABB. La programación de este robot angular se la realizará con el propio leguaje de la marca, denominado RAPID. Todos los movimientos que ejecutará el manipulador robótico podrán ser observados en el software RobotStudio antes de su implementación en el robot real.
- La comunicación entre el aplicativo de visión artificial con motor de ajedrez y el robot se dará a través de un “Socket”, en el que el servidor será la controladora del robot y el cliente un script de Python.

2.3. Justificación del proyecto

Los juegos de mesa físicos son un rico dominio de problemas para la investigación de la cooperación robot-humano, porque tales juegos tienen un grado de complejidad intermedia que puede ser gradualmente incrementada. Jugar juegos de mesa implica la percepción del tablero y de las piezas en el juego, la percepción del humano, razonando sobre el juego y el estado del mismo, y la manipulación de las piezas físicas mientras interactúa con el oponente. Por lo tanto, este proyecto debe entenderse como una antesala para proyectos de mayor envergadura, ya que, el mismo operará con una estructura general que permitirá su migración a otros usos y contextos.

Adicionalmente, la importancia de la visión artificial se ve reflejada en la cantidad de estudios que sobre ella recaen, de los que se pueden encontrar algunos como: desarrollo de algoritmos de control de calidad y control industrial; técnicas avanzadas de fabricación flexible y desensamblado automático; sensores para robots autónomos y detección de plásticos en alimentos.

Finalmente, la robótica en conjunto con: la visión artificial y la inteligencia artificial, juegan un papel medular en la industria en la actualidad, ya que, incrementan la capacidad de producción de la misma con solo un aumento en la inversión inicial. El uso de estas tecnologías brinda una gran flexibilidad a la hora de diseñar proyectos de automatización, dado que ofrece una amplia gama de posibilidades en lo que se refiera al trabajo y ejecución de tareas repetitivas, así como también tareas donde se requiere la toma de decisiones sin que sea necesario la supervisión de un operador.

3. Descripción

En este apartado se describirá las herramientas usadas para la elaboración del presente proyecto, para esto se tomó en cuenta dos aspectos: el software y el hardware. En el hardware o elementos físicos tenemos: computadora, robot, controlador, sensores, cámara, impresora 3D, servomotor, elemento terminal, tablero y piezas de ajedrez. En el lado del software encontramos los siguientes ítems: Python, OpenCV, PyCharm, Stockfish, RobotStudio, SolidWorks.

3.1. Descripción del hardware

Aquí se detallan todas las herramientas físicas utilizadas en la elaboración del proyecto, se presenta una breve descripción de cada elemento y como son utilizados para la creación del aplicativo. En primer lugar, el proyecto es desarrollado en un ordenador Dell G5, con sistema operativo Windows 10. En segundo lugar, se utiliza una cámara IP disponible en el laboratorio, para la obtención de imágenes. En tercer lugar, se imprime un elemento terminal que, en conjunto con un servomotor, servirá para la sujeción de las piezas de ajedrez. Por último, el manipulador utilizado es un robot IRB 140 que con ayuda de una controladora IRC5, harán los movimientos necesarios para el desarrollo de la partida de ajedrez.

3.1.1. Ordenador Dell G5

La programación tanto del sistema de visión artificial, así como también los movimientos que ejecutará el robot, corren sobre un ordenador portátil que cuenta con sistema operativo Windows 10, las características de conectividad que exige este proyecto no son altas, dado que solo se requiere un puerto de comunicación RJ-45 para la conexión de la cámara IP.

3.1.1.1. Características

Las principales características de este ordenador están descritas a continuación:

- Sistema operativo Windows 10 Home.
- Disco SSD PCIe NVMe de unidad doble, para una carga de aplicaciones muy rápida.
- Espesor de 25 mm, haciéndolo una portátil de fácil transporte.
- El chasis está construido en una aleación de magnesio y tiene un acabado de metal cepillado.
- Posee los siguientes puertos y ranuras: 1. Para micrófono/auriculares | 2. USB 3.1 SuperSpeed Gen 1 Type-A | 3. Puerto Thunderbolt 3 (USB 3.1 Gen 2 Type-C con compatibilidad para 40 Gb/s) | 4. HDMI 2.0 | 5. Pestaña para candado de seguridad Noble | 6. Alimentación | 7. Gigabit Ethernet RJ-45 | 8. USB 3.1 SuperSpeed Gen 1 Type-A | 9. Lector de tarjetas SD/MicroMedia 2 en 1.

3.1.1.2. Especificaciones

Las principales especificaciones de hardware del ordenador son las siguientes:

- Procesador Intel Core i7-8750H de 6 núcleos a 2.2GHz hasta 3.90GHz.
- 32 GB de RAM DDR4 | 1TB HDD + 256GB SSD.
- Pantalla IPS de 15.6 pulgadas 1920 x 1080.
- NVIDIA GeForce GTX 1050 Ti (4 GB GDDR5)
- Wi-Fi 802.11ac | Bluetooth 5.0.
- Thunderbolt 3 | HDMI.

- Windows 10 Home (64-Bit)

Figura 1. Portátil Dell G5 (“Portátil Dell G5 | Dell España,” n.d.)

3.1.2. Controlador IRC5

El controlador de un robot es el componente que procesa la información recolectada por los sensores, y según las instrucciones de las líneas de programación que han sido guardadas en el controlador, regula el movimiento de los actuadores (motores) u otros dispositivos de salida. El controlador maneja las fases de control, cinemática, dinámica, gestión de energía, comunicaciones y conexión de periféricos. Para este trabajo en concreto se utilizará el controlador IRC5 de ABB.

Figura 2. Controladora ABB IRC

Este controlador contiene los siguientes módulos:

- Módulo de accionamiento, que contiene el sistema de accionamiento.
- Control Module, que contiene el ordenador principal (incluidas cuatro ranuras PCI para tarjetas de extensión), el panel del operador, el interruptor principal, las interfaces de comunicación, la conexión para FlexPendant, los puertos de servicio y cierto espacio para los equipos del cliente, por ejemplo, tarjetas de E/S de ABB.
- El controlador también contiene el software de sistema, es decir RobotWare-OS, que incluye todas las funciones básicas de manejo y programación.

El panel de control del IRC5 está compuesto como se describen en la Figura 1Figura 3.

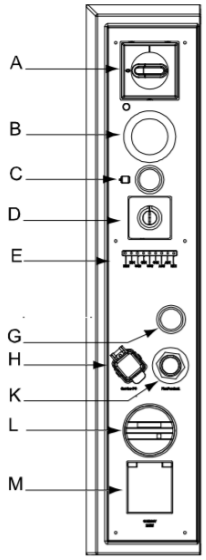


Figura 3. Panel de control (Robotics, 2004)

En la Tabla 1, se describe el funcionamiento de cada uno de los elementos que componen el panel de control.

Tabla 1. Panel de control (Robotics, 2004)

Pos	Nombre
A	Interruptor principal y control remoto de la alimentación de los módulos de accionamiento
B	Paro de emergencia. Si está introducido, tire para liberarlo.
C	MOTORS ON
D	Selector de modo de funcionamiento
E	LEDs de la cadena de seguridad (opción)
G	Pulsador de hot plug de FlexPendant
H	Conexión de PC de servicio
K	Conexión de FlexPendant
L	Contador de tiempo de funcionamiento (opción)
M	Toma de servicio a 115/230V, 200W (opcional)

3.1.3. Manipulador robótico ABB IRB140

El IRB 140 es un robot industrial de 6 ejes, con una carga útil de 6 kg y con un alcance de 810mm (con respecto al eje 5), diseñado específicamente para las industrias manufactureras que usan automatización flexible basada en robots. Se puede instalar en el suelo, de manera invertida o en la pared. El robot tiene una estructura abierta que está especialmente adaptada para un uso flexible, y puede comunicarse ampliamente con sistemas externos. Las dimensiones del robot se las puede apreciar en la Figura 4.

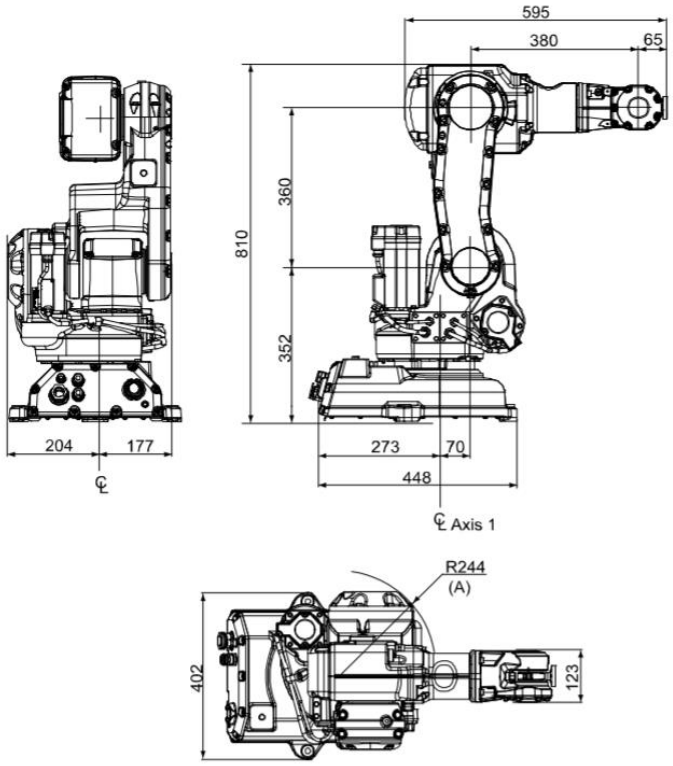


Figura 4. Dimensiones IRB 140 («IRB 140, M2004, Product specification», s. f.)

En lo que refiere a la movilidad del robot, el IRB 140 puede llegar a una amplia cantidad de puntos. El área de trabajo del manipulador está descrita en la

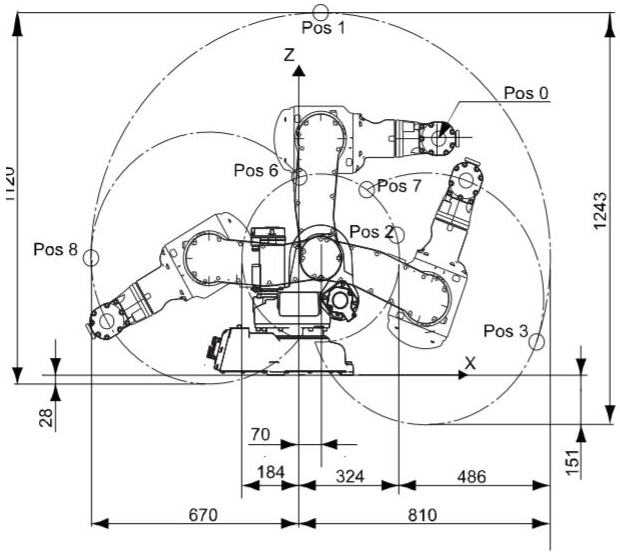


Figura 5. Límites de desplazamiento IRB 140(«IRB 140, M2004, Product specification», s. f.)

3.2. Descripción del software

3.2.1. Python

¿Qué es Python?

Python es un lenguaje de programación de alto nivel, interpretado y orientado a objetos, con semántica dinámica. Fue creado por Guido van Rossum y lanzado en 1991. Su alto nivel de estructuras de datos incorporadas, combinadas con la mecanografía dinámica y la encuadernación dinámica, lo hacen muy atractivo para el desarrollo rápido de aplicaciones, así como para su uso como lenguaje de scripts o de pegamento para conectar los componentes existentes entre sí. La sintaxis simple y fácil de aprender de Python enfatiza la legibilidad y por lo tanto reduce el costo de mantenimiento del programa. Python soporta módulos y paquetes, lo que fomenta el modularidad del programa y la reutilización del código. El intérprete de Python y la extensa biblioteca estándar están disponibles en forma de código fuente o binaria sin cargo para todas las plataformas principales, y pueden ser distribuidos libremente («What is Python? Executive Summary | Python.org», s. f.).

Figura 6. Python un lenguaje de programación de alto nivel

¿Qué puede hacer Python?

- Python puede ser usado en un servidor para crear aplicaciones web.
- Python puede ser usado junto con el software para crear flujos de trabajo.
- Python puede conectarse a sistemas de bases de datos. También puede leer y modificar archivos.
- Python puede ser usado para manejar grandes datos y realizar matemáticas complejas.
- Python puede ser usado para la creación rápida de prototipos, o para el desarrollo de software listo para la producción.

¿Por qué Python?

- Python funciona en diferentes plataformas (Windows, Mac, Linux, Raspberry Pi, etc.).
- Python tiene una sintaxis simple similar a la del idioma inglés.
- Python tiene una sintaxis que permite a los desarrolladores escribir programas con menos líneas que algunos otros lenguajes de programación.
- Python funciona en un sistema de interpretación, lo que significa que el código puede ser ejecutado tan pronto como se escribe. Esto significa que la creación de prototipos puede ser muy rápida.
- Python puede ser tratado de una manera procedimental, una manera orientada a objetos o una manera funcional.

Es bueno saber

- La versión principal más reciente de Python es Python 3, que usaremos en este proyecto. Sin embargo, Python 2, aunque no se actualiza con nada más que actualizaciones de seguridad, sigue siendo bastante popular.
- En este proyecto Python se escribirá en un editor de texto. Es posible escribir Python en un entorno de desarrollo integrado, como Thonny, Pycharm, Netbeans o Eclipse, que son especialmente útiles a la hora de gestionar grandes colecciones de archivos Python.

Sintaxis de Python comparada con otros lenguajes de programación

- Python fue diseñado para la legibilidad, y tiene algunas similitudes con el idioma inglés con influencia de las matemáticas.
- Python utiliza nuevas líneas para completar un comando, a diferencia de otros lenguajes de programación que a menudo utilizan punto y coma o paréntesis.
- Python se basa en la indentación, utilizando espacios en blanco, para definir el alcance; como el alcance de los bucles, las funciones y las clases. Otros lenguajes de programación suelen utilizar llaves para este fin. («Introduction to Python», s. f.)

3.2.2. PyCharm

¿Qué es PyCharm?

PyCharm es una plataforma híbrida desarrollada por JetBrains como un IDE, entorno de desarrollo integrado, por sus siglas en inglés, para Python. Se utiliza comúnmente para el desarrollo de aplicaciones en Python. Algunas de las plataformas de internet como: Twitter, Facebook, Amazon y Pinterest utilizan PyCharm como su IDE para Python. Soporta dos versiones: v2.x y v3.x.

Podemos ejecutar PyCharm en Windows, Linux o Mac OS. Además, contiene módulos y paquetes que ayudan a los programadores a desarrollar software usando Python en menos tiempo y con un mínimo esfuerzo. Además, también se puede personalizar según los requisitos de los desarrolladores.

Figura 7. PyCharm

A continuación, se he recopilado algunas de las características esenciales que ofrece PyCharm.

1. Editor de código inteligente:

- Ayuda a escribir códigos de alta calidad.

- Da esquemas de color para palabras clave, clases y funciones. Esto ayuda a aumentar la legibilidad y la comprensión del código.
- Ayuda a identificar errores fácilmente.
- Proporciona la función de autocompletar e instrucciones para completar el código.

2. Navegación por el código:

- Ayuda a los desarrolladores a editar y mejorar el código con menos esfuerzo y tiempo.
- Con la navegación por el código, un desarrollador puede navegar fácilmente a una función, clase o archivo.
- Un programador puede localizar un elemento, un símbolo o una variable en el código fuente en poco tiempo.
- Usando el modo de lente, además, un desarrollador puede inspeccionar y depurar a fondo todo el código fuente.

3. Refactorización

- Tiene la ventaja de hacer cambios eficientes y rápidos en las variables locales y globales.
- La refactorización en PyCharm permite a los desarrolladores mejorar la estructura interna sin cambiar el rendimiento externo del código.
- También ayuda a dividir clases y funciones más extendidas con la ayuda del método de extracción.

4. Asistencia para las Bibliotecas Científicas de Python

- PyCharm apoya las bibliotecas científicas de Python como Matplotlib, NumPy y Anaconda.
- Estas bibliotecas científicas ayudan a construir proyectos de Ciencia de Datos y Aprendizaje Automático.
- Consiste en gráficos interactivos que ayudan a los desarrolladores a entender los datos.
- Es capaz de integrarse con varias herramientas como IPython, Django, y Pytest. Esta integración ayuda a innovar soluciones únicas. («What is PyCharm? | What is Pycharm used for? | Intellipaat Blog», s. f.)

3.2.3. OpenCV

¿Qué es OpenCV?

OpenCV (Open Source Computer Vision Library) es una biblioteca de software de código abierto de visión por ordenador y aprendizaje automático. OpenCV se construyó para proporcionar una infraestructura común para las aplicaciones de visión por computador y para acelerar el uso de la percepción de la máquina en los productos comerciales. Al ser un producto con licencia BSD, OpenCV facilita a las empresas la utilización y modificación del código.

Figura 8. OpenCV

La biblioteca cuenta con más de 2500 algoritmos optimizados, que incluyen un conjunto completo de algoritmos de visión por computador y de aprendizaje automático, tanto clásicos como de última generación. Estos algoritmos pueden utilizarse para detectar y reconocer rostros, identificar objetos, clasificar acciones humanas en vídeos, seguir los movimientos de la cámara, rastrear objetos en movimiento, extraer modelos 3D de objetos, producir nubes de puntos 3D de cámaras estéreo, unir imágenes para producir una imagen de alta resolución de toda una escena, encontrar imágenes similares en una base de datos de imágenes, eliminar los ojos rojos de las imágenes tomadas con flash, seguir los movimientos de los ojos, reconocer el paisaje y establecer marcadores para superponerlo con la realidad aumentada, etc. OpenCV cuenta con más de 47 mil personas de la comunidad de usuarios y se estima que el número de descargas supera los 18 millones. La biblioteca se utiliza ampliamente en empresas, grupos de investigación y organismos gubernamentales.

Junto con empresas bien establecidas como Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota que emplean la biblioteca, hay muchas empresas de nueva creación como Applied Minds, VideoSurf y Zeitera, que hacen un uso extensivo de OpenCV. Los usos desplegados de OpenCV abarcan desde la unión de imágenes de la calle, la detección de intrusiones en los vídeos de vigilancia en Israel, la supervisión de equipos de minas en China, la ayuda a los robots para navegar y recoger objetos en el garaje de Willow, la detección de accidentes de ahogamiento en piscinas en Europa, la realización de arte interactivo en España y Nueva York, el control de las pistas de aterrizaje en busca de escombros en Turquía, la inspección de las etiquetas de los productos en fábricas de todo el mundo hasta la detección rápida de rostros en Japón.

Tiene interfaces C++, Python, Java y MATLAB y es compatible con Windows, Linux, Android y Mac OS. OpenCV se inclina principalmente hacia aplicaciones de visión en tiempo real y aprovecha las instrucciones MMX y SSE cuando están disponibles. En este momento se están desarrollando activamente las interfaces CUDA y OpenCL. Hay más de 500 algoritmos y unas 10 veces más funciones que componen o soportan esos algoritmos. OpenCV está escrito de forma nativa en C++ y tiene una interfaz planificada que funciona perfectamente con los contenedores STL (Howse & Michino, 2020).

3.2.4. Stockfish

Stockfish es un motor de ajedrez gratuito y de código abierto, disponible para varias plataformas de escritorio y móviles. Es desarrollado por Marco Costalba, Joona Kiiski, Gary Linscott, Stéphane Nicolet, y Tord Romstad, con muchas contribuciones de una comunidad de desarrolladores de código abierto.

Stockfish está constantemente clasificado en el primer lugar o cerca del primer lugar de la mayoría de las listas de clasificación de motores de ajedrez y es el motor de ajedrez convencional más fuerte del mundo. Ganó los campeonatos mundiales no oficiales de ajedrez informático en las temporadas 6 (2014), 9 (2016), 11 (2018), 12 (2018), 13 (2018), 14 (2019) y 16 (2019). Terminó subcampeón en la temporada 5 (2013), 7 (2014), 8 (2015) y 15 (2019).

Figura 9. Stockfish

Características

Stockfish puede usar hasta 512 hilos de CPU en sistemas multiprocesadores. El tamaño máximo de su tabla de transposición es de 128 GB. Stockfish implementa una búsqueda alfa-beta avanzada y utiliza bitboards. En comparación con otros motores, se caracteriza por su gran profundidad de búsqueda, debido en parte a una poda más agresiva, y a la reducción de los movimientos tardíos.

Stockfish soporta Chess960, que es una de las características que se heredó de Glaurung.

Plataformas

Las versiones de lanzamiento y desarrollo están disponibles como código fuente C++ y como versiones precompiladas para Microsoft Windows, macOS, Linux 32-bit/64-bit y Android.

Stockfish ha sido un motor muy popular para varias plataformas. En el escritorio, es el motor de ajedrez por defecto, junto con los programas de interfaz del Club de Ajedrez de Internet BlitzIn y Dasher. En la plataforma móvil, ha sido combinado con la aplicación Stockfish, SmallFish y Droidfish. Otras interfaces gráficas de usuario compatibles con Stockfish incluyen Fritz, Arena, Stockfish para Mac, y PyChess («About - Stockfish - Open Source Chess Engine», s. f.).

3.2.5. RobotStudio

¿Qué es RobotStudio?

Es un software de programación y simulación fuera de línea de ABB. La capacidad de programar un robot en el mundo virtual antes de que opere en el mundo real ha cambiado drásticamente la forma en que las empresas y los individuos piensan en la programación de los robots. En la última década se ha convertido en una forma cada vez más popular de probar el funcionamiento de los robots antes de que un error en la fábrica provoque daños, paradas y/o pérdidas de dinero. El método tradicional de programar robots, usando un FlexPendant conectado al controlador del robot, funciona bien para algunas tareas, pero los robots han sido colocados en operaciones cada vez más intrincadas y

complicadas, cosas que incluso el programador humano más hábil que mira una pantalla llena de innumerables líneas de código sería difícil de lograr.

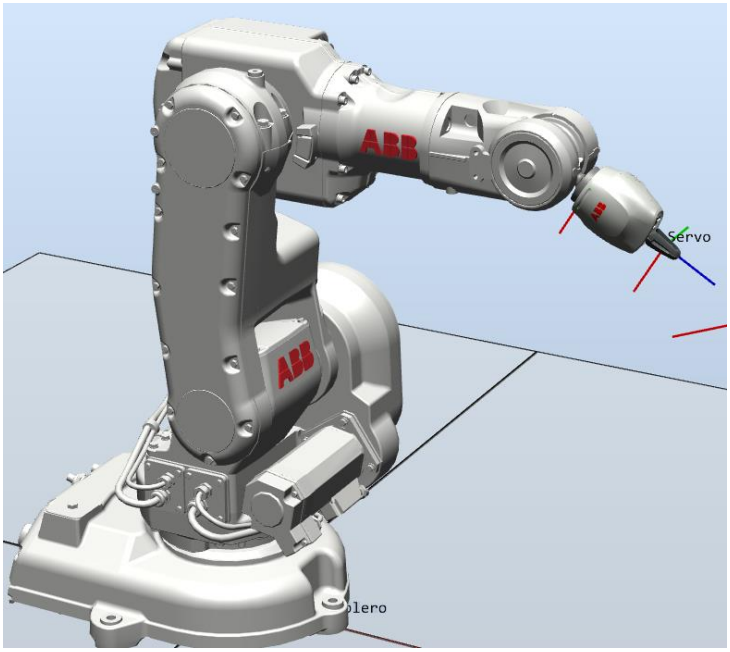


Figura 10. RobotStudio

RobotStudio elimina las conjeturas de la programación. En el mundo virtual tridimensional basado en PC, cada parte de una operación o proceso puede ejecutarse y visualizarse sin necesidad de pedir una pieza física real o un robot. Una vez que el programa se completa en el mundo virtual, puede simplemente descargarse directamente al controlador del robot en el mundo real, y siempre que todo en el mundo real esté configurado exactamente como estaba en el mundo virtual, el programa se ejecutará exactamente como lo hizo en el PC («RobotStudio - ABB Robotics», s. f.).

Características de RobotStudio

- La programación puede hacerse en la oficina sin necesidad de detener la producción en la fábrica.
- Los programas pueden ser preparados con antelación.
- El entrenamiento y la optimización pueden hacerse sin perturbar la producción.
- Se reduce el riesgo de daños o retrasos costosos.
- La instalación y puesta en marcha de nuevos sistemas es más rápida.
- El cambio entre las series de producción es más rápido.
- La productividad se incrementa enormemente.

3.2.6. SolidWorks

SOLIDWORKS, un sólido programa de diseño e ingeniería asistida por ordenador de modelado, es una de las opciones de software más populares para los ingenieros y diseñadores hoy en día.

Figura 11. SolidWorks

SOLIDWORKS fue desarrollado por el graduado del MIT Jon Hirschtick y fue comprado por Dassault Systems en 1997. El software ahora abarca un número de programas que pueden ser usados tanto para diseño 2D como 3D.

SOLIDWORKS se utiliza para desarrollar sistemas mecatrónicos de principio a fin. En la etapa inicial, el software se utiliza para la planificación, la ideación visual, el modelado, la evaluación de la viabilidad, la creación de prototipos y la gestión de proyectos. El software se utiliza luego para el diseño y la construcción de elementos mecánicos, eléctricos y de software. Por último, el software puede utilizarse para la gestión, incluida la gestión de dispositivos, el análisis, la automatización de datos y los servicios de nube.

Las soluciones de software de SOLIDWORKS son utilizadas por los ingenieros mecánicos, eléctricos y electrónicos para formar un diseño conectado. El conjunto de programas tiene como objetivo mantener a todos los ingenieros en comunicación y capaces de responder a las necesidades o cambios de diseño.

Una muestra de los productos que forman parte de SOLIDWORKS, como se describen en su sitio web, incluyen:

- CircuitWorks: un traductor electrónico CAD/ECAD que permite a los ingenieros crear modelos 3D precisos de placas de circuitos.
- CAM: un complemento para todas las versiones de SOLIDWORKS CAD que le permite preparar sus diseños para la fabricación en una fase temprana del ciclo de desarrollo.
- Electrical 3D: le permite colocar componentes eléctricos y utilizar la tecnología de enrutamiento de SOLIDWORKS para interconectar automáticamente los elementos de diseño eléctrico dentro de un modelo 3D. Los esquemas 2D y los modelos 3D se sincronizan bidireccionalmente en tiempo real, de modo que cualquier cambio se actualiza automáticamente.
- Simulation: utiliza el análisis de elementos finitos (FEA) para predecir el comportamiento físico de un producto en el mundo real, probando virtualmente los modelos CAD.
- Visualize: aprovecha tus datos CAD 3D para crear contenido de calidad fotográfica de la forma más rápida y sencilla posible, desde imágenes hasta animaciones, contenido web interactivo y realidad virtual inmersiva.

SOLIDWORKS continúa adaptando sus soluciones para incluir nuevas capacidades basadas en los comentarios de los usuarios. SOLIDWORKS 2020 presenta una serie de mejoras, como un mejor rendimiento, flujos de trabajo racionalizados y 3DExperience, una plataforma basada en la nube («What is SOLIDWORKS? | Capitol Technology University», s. f.).

4. Metodología

En este capítulo se ilustra el desarrollo del aplicativo que permitirá la realización de partidas de ajedrez hombre-máquina, mediante el uso de un brazo robótico y el uso de visión artificial por computadora. Para la consecución de este objetivo, se divide a este proyecto en los siguientes apartados: 1) Procesamiento de imágenes, 2) Motor de ajedrez, 3) Interfaz de comunicación, 4) Interfaz gráfica de usuario, 5) Software de la unidad central de control, 6) Programación de movimientos del robot ABB IRB 140, y 7) Diseño de un elemento terminal para la sujeción de piezas de ajedrez. Cada uno de estos puntos detallados anteriormente cuenta con una breve introducción y la explicación del trabajo necesario realizado para el correcto funcionamiento del aplicativo. En la Figura 12 , se observa un diagrama de funcionamiento del aplicativo.

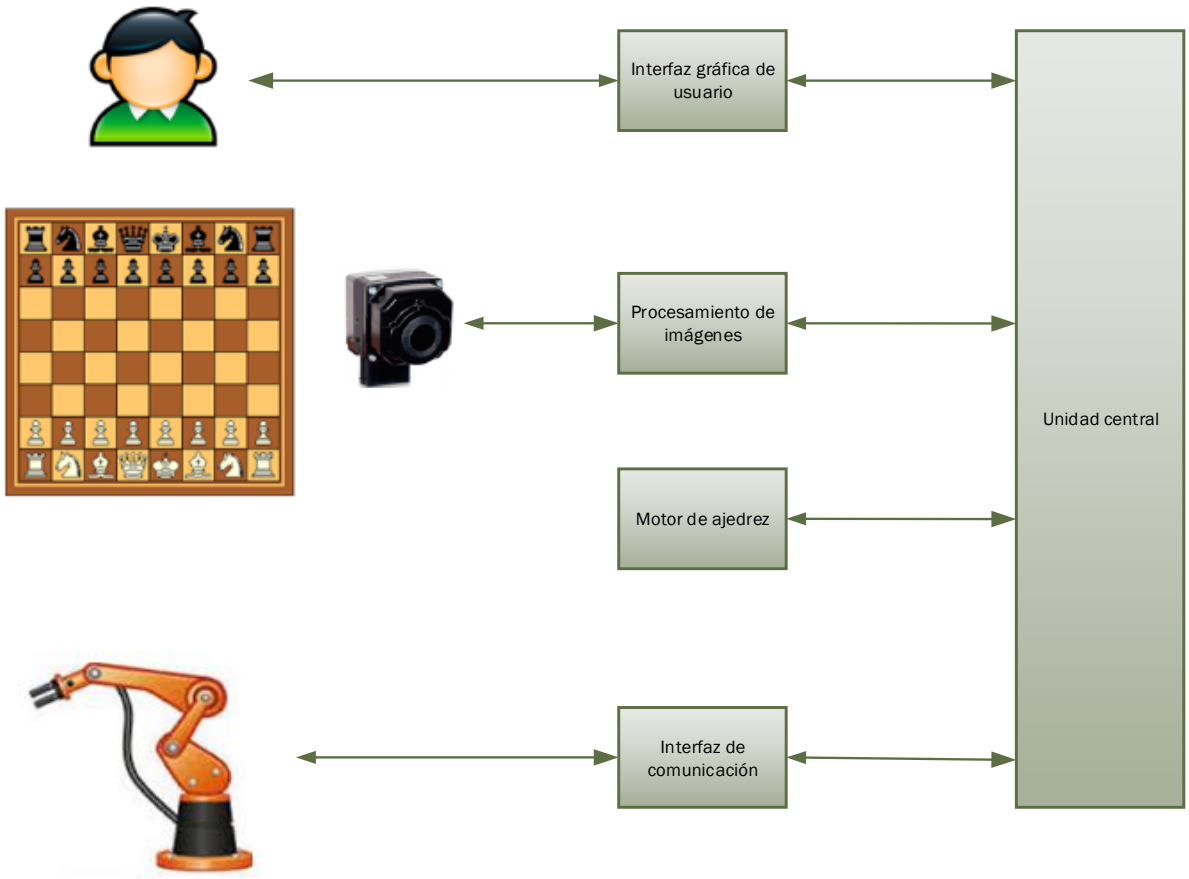


Figura 12. Breve descripción de la arquitectura del sistema juego autónomo de ajedrez

4.1. Procesamiento de imágenes

Este apartado contiene dos partes fundamentales del sistema, los cuales son: el algoritmo de reconocimiento de tablero de ajedrez y el algoritmo de detección de movimiento de las piezas. Cada uno de estos cumple una función específica. El primero, es tener un mapa de bits de la imagen original del tablero, es decir, el tablero de ajedrez sin ninguna pieza. Esto asignara a cada casilla o escaque un nombre, como se observa en la Figura 13. El segundo, utiliza dos imágenes, una previa y una actual, del tablero de ajedrez. El algoritmo hace una comparación de ambas fotografías y determina los cambios en el color de la imagen, para establecer si hubo o no algún movimiento. Ambos algoritmos se detallarán con mayor profundidad a continuación.

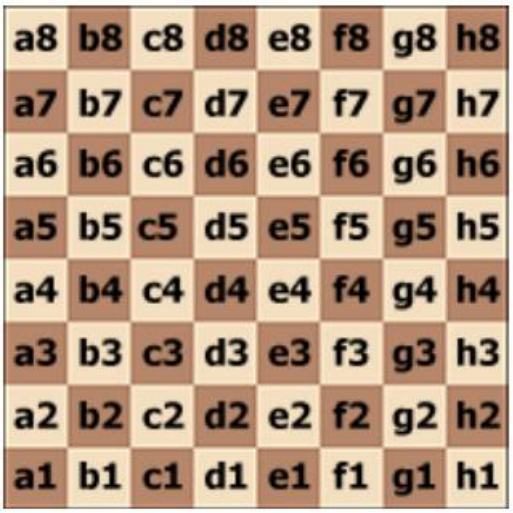


Figura 13. Notación algebraica de las casillas de un tablero de ajedrez (*Leyes del Ajedrez de la FIDE*, 2017)

4.1.1. Algoritmo de reconocimiento de tablero de ajedrez

El objetivo con respecto al procesamiento del tablero de ajedrez es encontrar un mapa de la imagen original a un tablero de ajedrez ideal. Tener este mapeo simplifica el análisis del color de cada cuadrado, para después asociar cada cuadrado, que en el futuro se le denominará escaque, una pieza de ajedrez. En detalle, el proceso de reconocimiento de tablero consiste en de los siguientes pasos:

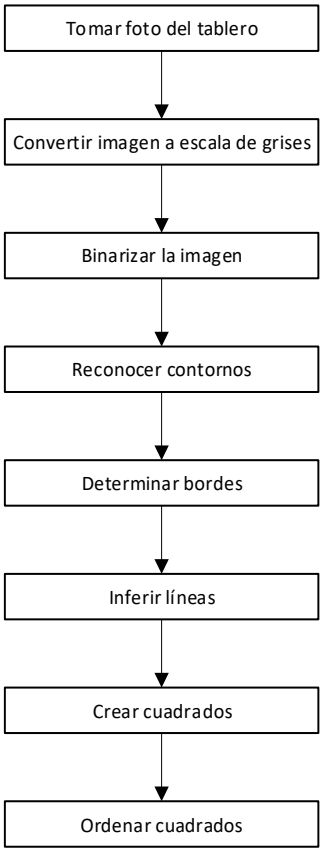


Figura 14. Diagrama algoritmo de reconocimiento de tablero

4.1.1.1. Toma de foto del tablero

Para la adquisición de una foto del tablero, se utilizará una función de OpenCV llamada *VideoCapture().* La sintaxis de esta función es la siguiente:

<VideoCapture object> = cv.VideoCapture(index)

Parámetros

- **index:** Se coloca el id del dispositivo de captura de vídeo a abrir. Para abrir la cámara predeterminada, solo se coloca 0, si se necesita abrir una cámara IP. Se coloca la dirección de la misma en el lugar de index («OpenCV: cv::VideoCapture Class Reference», s. f.).

En la Figura 15, se aprecia la obtención de la imagen mediante la cámara del dispositivo móvil.

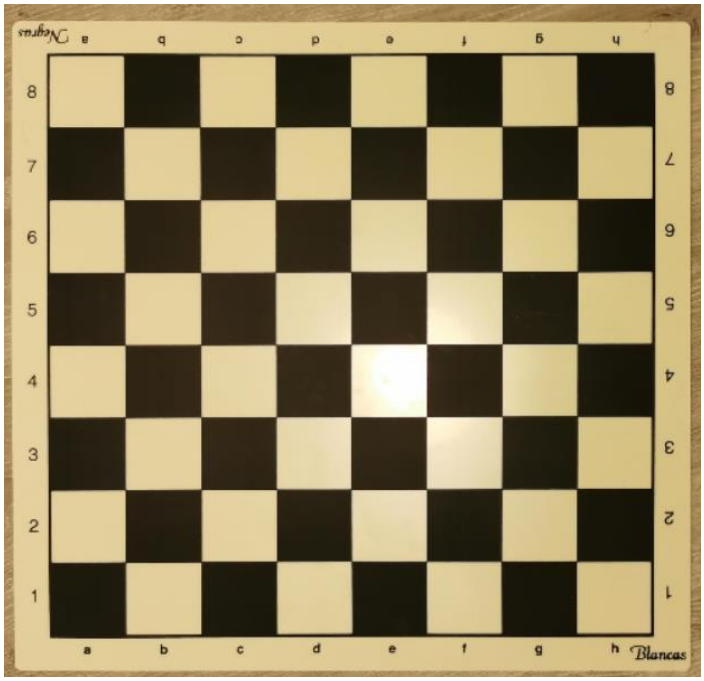


Figura 15. Fotografía del tablero de ajedrez

4.1.1.2. Convertir imagen a escala de grises

Después de obtenida la imagen a color del tablero, se procede a transformarla en una a escala de grises. Para ello, se utilizará el siguiente método de OpenCV: *cv2.cvtColor()*. Este método se utiliza para convertir una imagen de un espacio de color a otro. Hay más de 150 métodos de conversión de espacios de color disponibles en OpenCV. La sintaxis de este método es la siguiente:

cv2.cvtColor(src, code[, dst[, dstCn]])

Parámetros:

- **src:** Es la imagen cuyo espacio de color debe ser cambiado.
- **code:** Es el código de conversión del espacio de color.
- **dst**: Es la imagen de salida del mismo tamaño y profundidad que la imagen src. Es un parámetro opcional.

- **dstCn**: Es el número de canales en la imagen de destino. Si el parámetro es 0, entonces el número de canales se deriva automáticamente de src y del código. Es un parámetro opcional («Python OpenCV | cv2.cvtColor() method - GeeksforGeeks», s. f.).

En la Figura 16 se observa el cambio de espacio de color *BGR* a *GRAY*, utilizando la siguiente instrucción:

gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

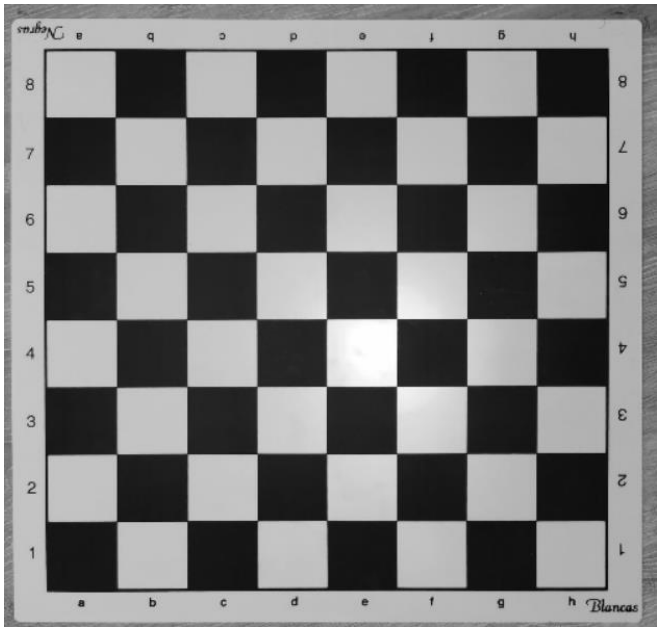


Figura 16. Tablero de ajedrez en escala de grises

4.1.1.3. Binarización de la imagen

Una vez conseguida la imagen en escala de grises se procede a binarizarla. Para este paso, se utiliza el método *AdaptiveThreshold()*. En el simple Thresholding, se utilizó un valor global de umbral que permaneció constante en todo momento. Por lo tanto, un valor de umbral constante no ayudará en el caso de condiciones de iluminación variables en diferentes áreas. El umbral adaptativo (Adaptive Thresholding) es el método en el que el valor umbral se calcula para regiones más pequeñas. Esto conduce a diferentes valores umbral para diferentes regiones con respecto al cambio de iluminación. La sintaxis de este método es la siguiente:

cv2.adaptiveThreshold(source, maxVal, adaptiveMethod, thresholdType, blocksize, constant)

Parámetros:

- **source:** Matriz de imágenes de entrada (un solo canal, 8 bits o punto flotante)
- **maxVal**: Valor máximo que se puede asignar a un píxel.
- **AdaptiveMethod:** El método adaptativo decide cómo se calcula el valor umbral.

cv2.ADAPTIVE_THRESH_MEAN_C: *Threshold Value* = (Media de los valores de la zona de vecindad - valor constante). En otras palabras, es la media del vecindario *blockSize×blockSize* de un punto menos constante.

cv2.ADAPTIVE_THRESH_GAUSSIAN_C: *Threshold Value* = (Suma ponderada por Gauss de los valores de la vecindad - valor constante). En otras palabras, es una suma ponderada del vecindario blockSize×blockSize de un punto menos constante.

- **threshodType:** El tipo de umbral a aplicar.
- **blockSize:** Tamaño de un vecindario de píxeles que se utiliza para calcular un valor umbral.
- **constant:** Un valor constante que se resta de la media o suma ponderada de los píxeles del vecindario («Python | Thresholding techniques using OpenCV | Set-2 (Adaptive Thresholding) - GeeksforGeeks», s. f.).

Se puede observar en la Figura 17 como todos los píxeles se han transformado en blancos o negros, esto facilitará los siguientes pasos. Para este caso se utilizó la siguiente instrucción:

adaptiveThresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 125, 1)

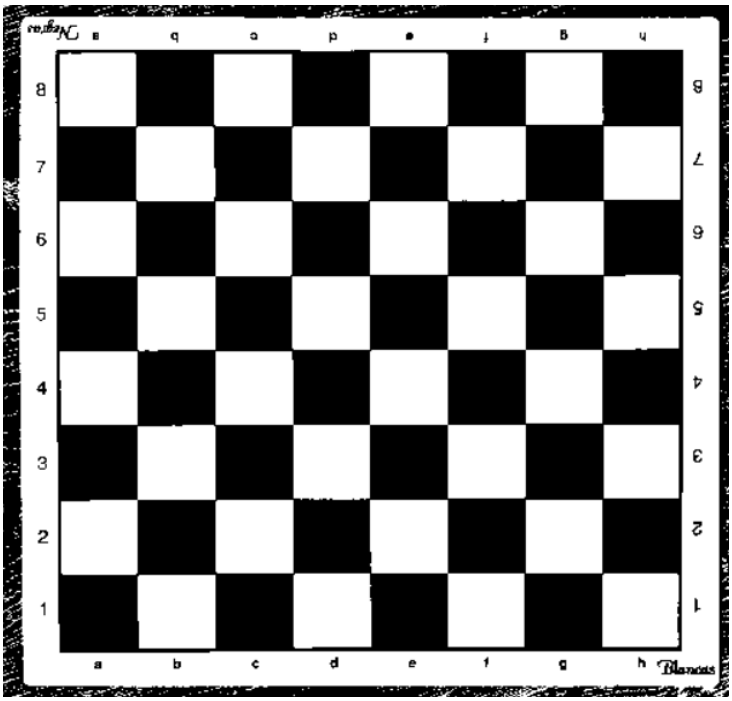


Figura 17. Tablero de ajedrez binarizado

4.1.1.4. Reconocer bordes

Después de conseguida la imagen binarizada, se procede a reconocer los bordes de la misma. Para esto se utilizará el método *Canny().* Este método tiene los siguientes algoritmos: 1) Reducción de ruido, 2) Encontrar el gradiente de intensidad de la imagen, 3) Supresión de píxeles no deseados, 4) Histéresis de umbral, decisión de cuáles son bordes y cuáles no. La sintaxis de este método es la siguiente:

cv2.Canny(image, threshold1, threshold2[, edges[, apertureSize[, L2gradient]]])

Parámetros:

- **image:** imagen de entrada de 8 bits de un solo canal.

- **edges:** mapa de bordes de salida; tiene el mismo tamaño y tipo que la imagen.
- **threshold1:** - primer umbral para el procedimiento de histéresis.
- **thresohold2:** segundo umbral para el procedimiento de histéresis.
- **apertureSize:** tamaño de apertura para el operador Sobel().
- **L2gradient:** una bandera, que indica si una norma L_2 más precisa $=\sqrt{(dI/dx)^2 + (dI/dy)^2}$ debe usarse para calcular la magnitud del gradiente de la imagen (L_2gradient = true), o si la norma L_1 por defecto $= |dI/dx| + |dI/dy|$ es suficiente (L_2gradient = false).

La función encuentra los bordes en la imagen de entrada y los marca en los bordes del mapa de salida utilizando el algoritmo de Canny. El valor más pequeño entre el *threshold1* y el *threshold2* se utiliza para la vinculación de los bordes. El valor más grande se usa para encontrar segmentos iniciales de bordes fuertes («Canny Edge Detection — OpenCV-Python Tutorials 1 documentation», s. f.).

En la Figura 18 se aprecia los bordes obtenidos después utilizar el algoritmo de Canny.

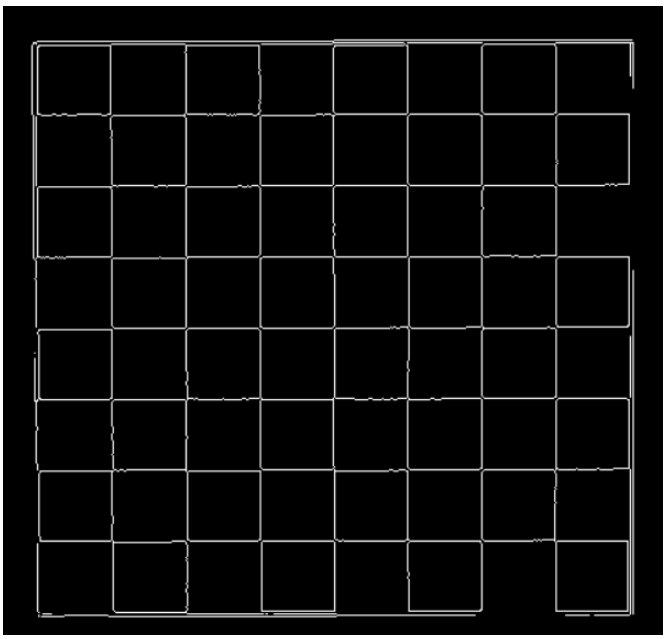


Figura 18. Bordes del tablero de ajedrez

4.1.1.5. Inferir líneas

Una vez obtenida los bordes del tablero de ajedrez, se procede a inferir líneas en base a los bordes obtenidos. Para la obtención de las líneas se utilizará la Transformada de Hough. Esta transformada permite detectar cualquier tipo de forma, si se puede representar esta forma de manera matemática, el algoritmo podrá detectar la forma incluso si está distorsionada un poco o rota. Para este caso en específico utilizaremos la transformada de Hough probabilística. La Transformación probabilística de Hough es una optimización de la Transformada de Hough. No toma en cuenta todos los puntos, sino que toma sólo un subconjunto aleatorio de puntos y eso es suficiente para la detección de líneas. Sólo tenemos que disminuir el umbral. La sintaxis de este método es la siguiente:

cv2.HoughLinesP(image, rho, theta, threshold[, lines[, minLineLength[, maxLineGap]]])

Parámetros:

- **Image**: 8-bit, imagen de fuente binaria de un solo canal. La imagen puede ser modificada por la función.
- **lines:** Vector de salida de líneas. Cada línea está representada por un vector de 4 elementos $(x1, y1, x2, y2)$, donde $(x1, y1)$ y $(x2, y2)$ son los puntos finales de cada segmento de línea detectado.
- **Rho:** Resolución de distancia del acumulador en píxeles.
- **Theta:** Resolución del ángulo del acumulador en radianes.
- **Threshold**: Parámetro de umbral del acumulador. Sólo se devuelven aquellas líneas que obtienen suficientes votos ($> threshold$).
- **minLineLength**: Longitud mínima de la línea. Los segmentos de línea más cortos que eso son rechazados.
- **maxLineGap:** Máxima brecha permitida entre puntos de la misma línea para unirlos.

En la Figura 19 , se aprecia como se dibujan las líneas en el lugar donde se encontraban los bordes. Estas líneas serán utilizadas para después encontrar esquinas.

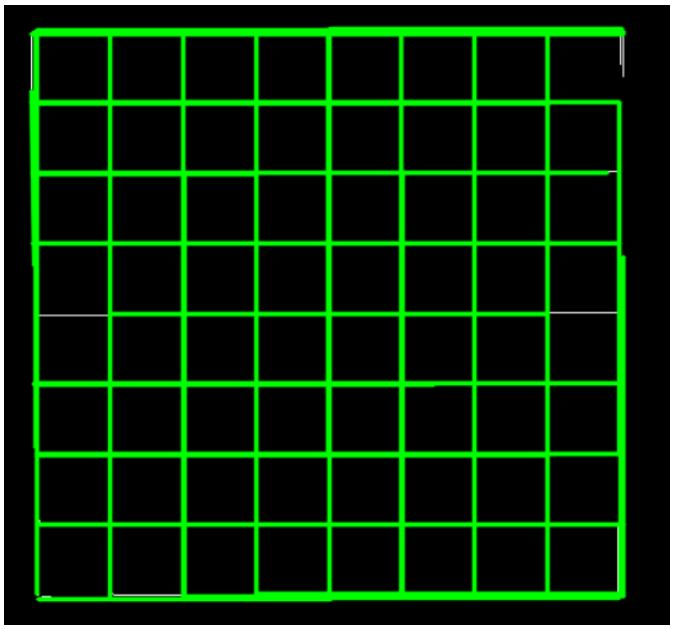


Figura 19. Líneas de tablero de ajedrez

4.1.1.6. Crear cuadrados

Para la creación de los cuadrados del tablero, lo primero que se procede a realizar es encontrar las esquinas de los mismos. Para ello se encontrarán los puntos en donde se intersecan las líneas. Se tratará a las líneas en su forma matemática, para de esta manera hallar el determinante de las mismas, que en este caso serán las coordenadas de las esquinas. El código para hallar el determinante en este caso es el siguiente:

```python
def find_intersection(self,other):
    '''
    Encuentra la intersección de esta línea y otra. Una línea debe ser horizontal
    y la otra debe ser vertical.
    '''

    # Determinante para hallar los puntos de intersección de las líneas
    x = ((self.x1*self.y2 - self.y1*self.x2)*(other.x1-other.x2) - (self.x1-self.x2)*(other.x1*other.y2 - other.y1*other.x2))/ \
        ((self.x1-self.x2)*(other.y1-other.y2) - (self.y1-self.y2)*(other.x1-other.x2))
    y = ((self.x1*self.y2 - self.y1*self.x2)*(other.y1-other.y2) - (self.y1-self.y2)*(other.x1*other.y2 - other.y1*other.x2))/ \
        ((self.x1-self.x2)*(other.y1-other.y2) - (self.y1-self.y2)*(other.x1-other.x2))
    x = int(x)
    y = int(y)

    return x,y
```

Figura 20. Función para encontrar la intersección de líneas

Después de esto se eliminará las esquinas duplicadas, observando que la distancia entre ellas sea mayor a 10, sino lo es, la esquina se eliminara, caso contrario dibujará las esquinas con círculos. Para esto utilizamos la siguiente función:

```python
def findCorners (self, horizontal, vertical, colorEdges):
    '''
    Encuentra esquinas en la intersección de líneas horizontales y verticales
    '''

    # Encuentra esquinas (intersecciones de líneas).
    corners = []
    for v in vertical:
        for h in horizontal:
            s1,s2 = v.find_intersection(h)
            corners.append([s1,s2])

    # Elimina esquinas duplicadas
    dedupeCorners = []
    for c in corners:
        matchingFlag = False
        for d in dedupeCorners:
            if math.sqrt((d[0]-c[0])*(d[0]-c[0]) + (d[1]-c[1])*(d[1]-c[1])) < 20:
                matchingFlag = True
                break
        if not matchingFlag:
            dedupeCorners.append(c)

    for d in dedupeCorners:
        cv2.circle(colorEdges, (d[0],d[1]), 10, (0,0,255))
```

Figura 21. Función para encontrar esquinas

En la Figura 22, se puede observar en círculos rojos las esquinas creadas por el aplicativo.

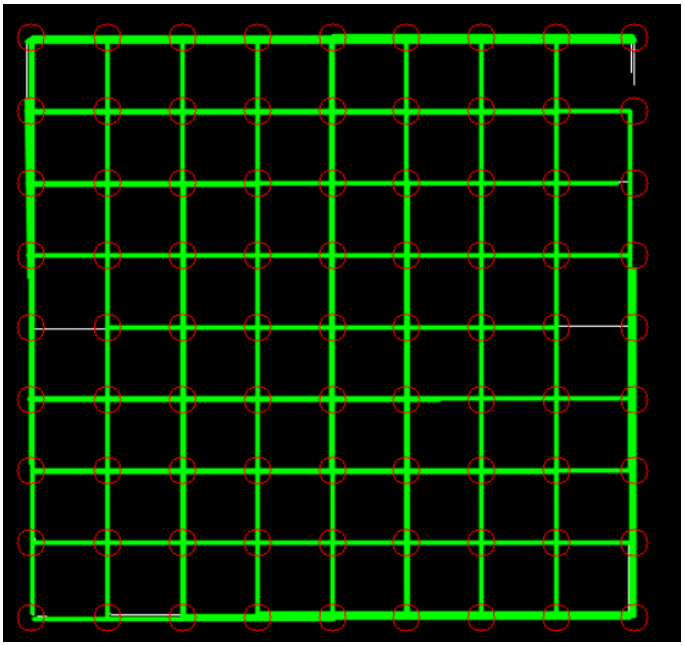


Figura 22. Esquinas encontradas en el tablero de ajedrez

Seguido de esto, hay que dibujar los cuadrados, para esto, se ordena las esquinas por fila y por columna, como se describe a continuación:

```
# Ordena esquinas por fila
corners.sort(key=lambda x: x[0])
rows = [[],[],[],[],[],[],[],[],[]]
r = 0
for c in range(0, 81):
    if c > 0 and c % 9 == 0:
        r = r + 1

    rows[r].append(corners[c])

letters = ['a','b','c','d','e','f','g','h']
numbers = ['1','2','3','4','5','6','7','8']
Squares = []

# Ordena esquinas por columna
for r in rows:
    r.sort(key=lambda y: y[1])
```

Figura 23. Clasificación de esquinas por fila y columna

Por último, se dibujará todos los cuadrados y se ordenará siguiendo la notación de un tablero de ajedrez, que consiste en: 1) Letras para columnas y 2) números para filas. Todos los 64 escaques del tablero de ajedrez quedarán organizados como se observa en la

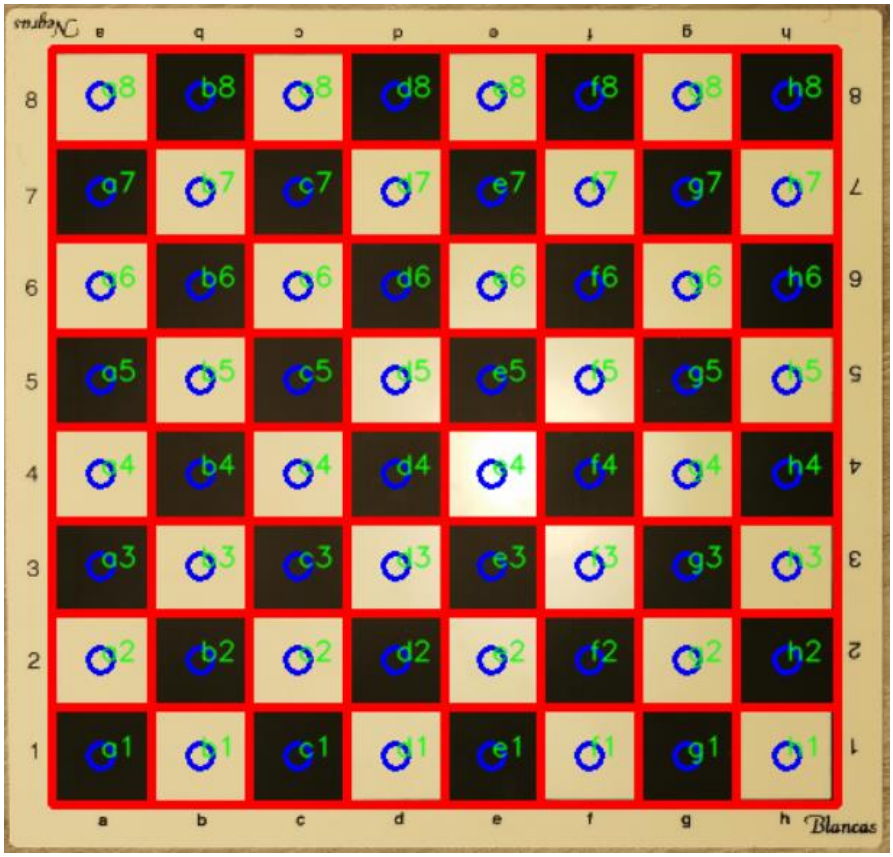


Figura 24. Clasificación de escaques del tablero de ajedrez

4.1.2. Algoritmo de detección de movimiento de piezas

El objetivo de este apartado es detectar los movimientos efectuados por el jugador humano, para que de esta manera y con la ayuda de un motor de ajedrez, en este caso, Stockfish, sea reconocida la jugada y se interprete qué tipo de movimiento fue: 1) movimiento normal, 2) movimiento de captura, 3) movimiento de enroque, 4) movimiento de captura al paso y 5) movimiento de coronación de un peón. Para después, devolver un movimiento que será ejecutado por el brazo robótico. El algoritmo de detección de movimiento de piezas se detalla en el siguiente diagrama de flujo.

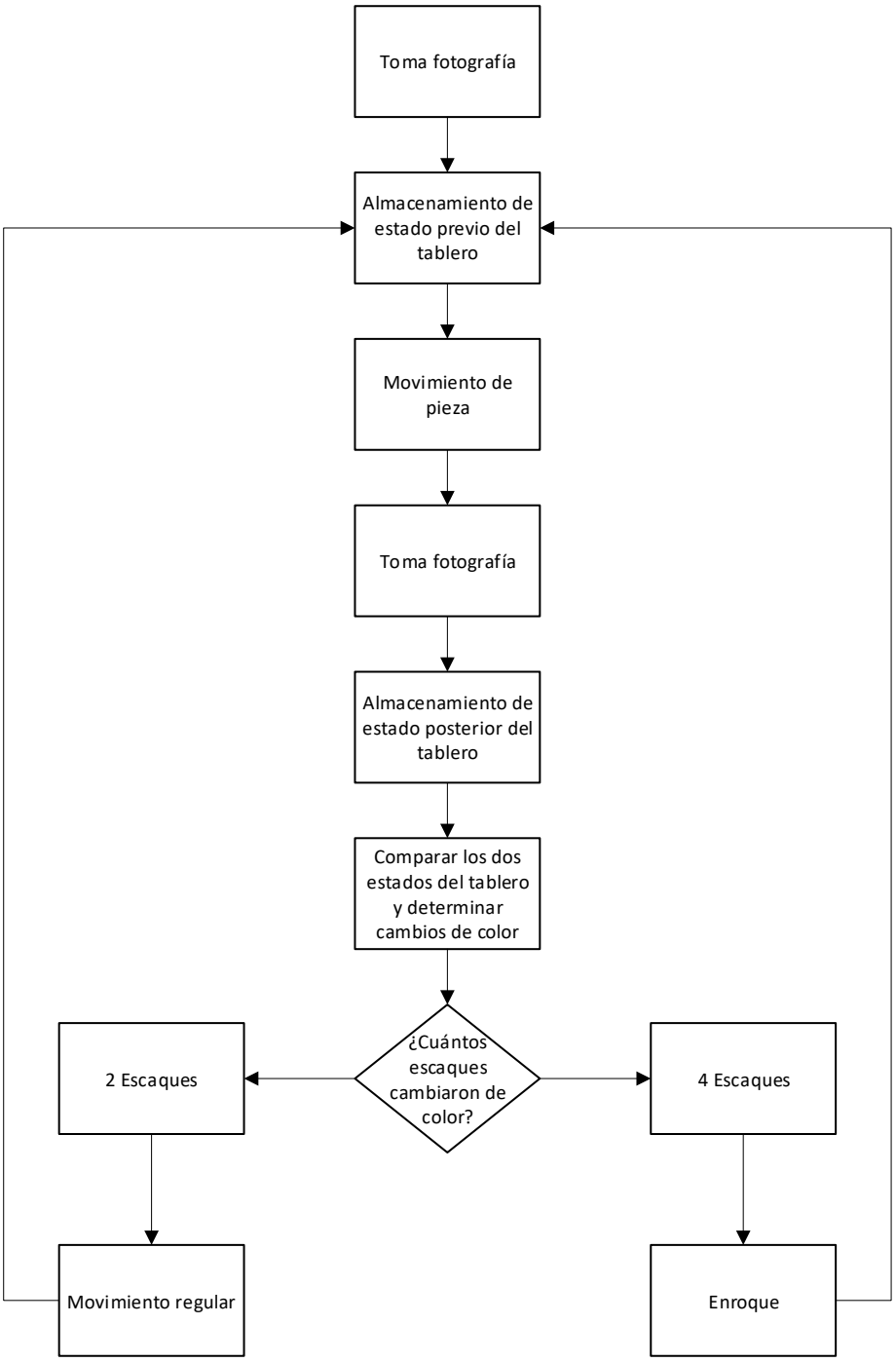
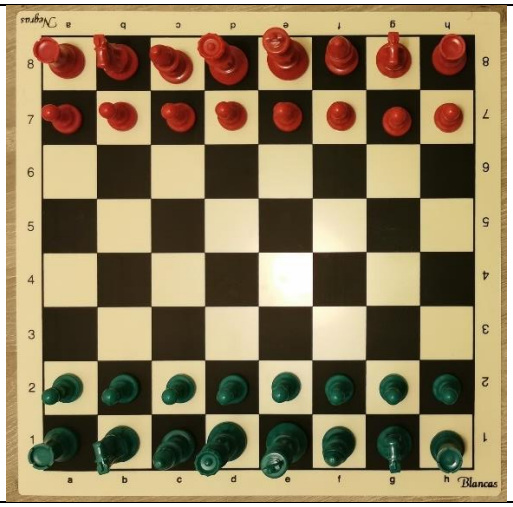
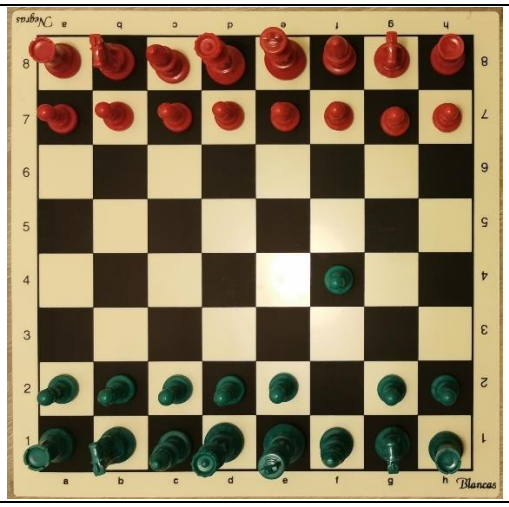


Figura 25. Algoritmo de detección de movimiento de piezas

1. Este algoritmo requiere dos fotos: una antes de un movimiento y una después. Así, después de que un movimiento es hecho, el usuario da clic en <<Hecho>>, diciéndole a la cámara que tome una imagen que se pasa al algoritmo. En la Tabla 1Tabla 2, se observa dos fotografías en las que está el estado anterior y el estado actual del tablero.

Tabla 2. Estados previo y actual del tablero de ajedrez

Anterior	Actual
	

2. A continuación, el programa itera a través de todas las casillas del tablero, determinando la diferencia en el valor de color de la fotografía anterior a la actual. La diferencia o distancia entre dos colores es una métrica que permite un examen cuantificado de lo que antes solo se podía describir con adjetivos.

   Como la mayoría de las definiciones de diferencia de color son distancias dentro de un espacio de color, el medio estándar para determinar las distancias es la distancia euclidiana. Si uno tiene actualmente una tupla RGB (Rojo, Verde, Azul) y desea encontrar la diferencia de color, computacionalmente una de las más fáciles es llamar a las dimensiones lineales R, G, B que definen el espacio de color.

$$distancia = \sqrt{(R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2}$$

   Se colocará un umbral lo suficientemente grande, con el cual se observará un cambio de color significativo cuando se ha realizado un movimiento y no solo una diferencia causada por cambios en la iluminación del medio exterior. Para determinar un valor adecuado de diferencia de color se utilizará la Tabla 3.

Tabla 3. Escala típica de cambio en los valores de color de 0 a 100 (Mokrzycki & Tatol, 2011)

Distancia Euclidiana	Percepción
<= 1.0	No es perceptible para los ojos humanos,
1 – 2	Perceptible a través de una observación cercana.
2 – 10	Perceptible a simple vista
11 – 49	Diferencia es claramente notable
100	Color totalmente opuesto

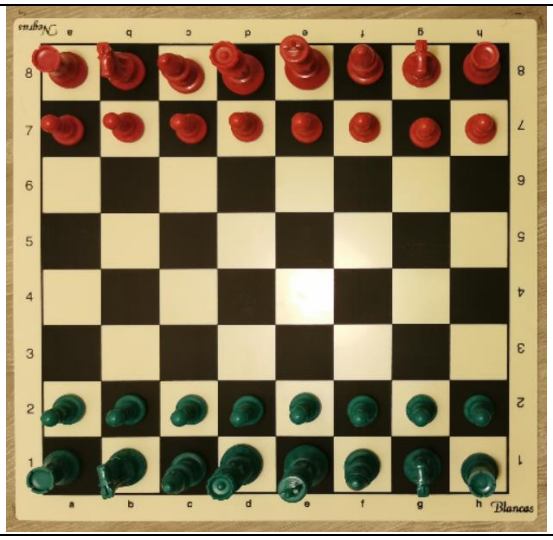
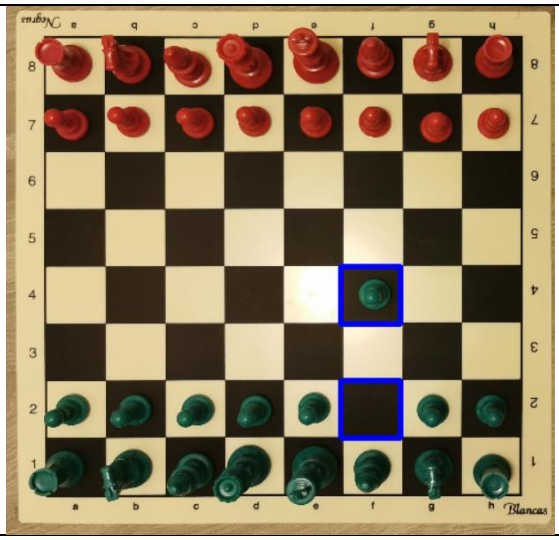
El valor que se tomó fue de 25, después de haber realizado varias pruebas. Por consiguiente, en un movimiento típico, dos cuadrados tienen el mayor cambio en el color, el cuadrado desde que se movió una pieza y el cuadrado al que se movió.

3. Para determinar qué tipo de acción realizó el jugador humano, se toma en cuenta los siguientes casos:

- Si no hay diferencia entre el estado previo y el actual, esto significa que no hay ningún cambio en el juego. Por esta razón, la diferencia de color sobre el tablero no es un movimiento normal.
- Si sólo hay una casilla ocupada y una no ocupada con diferencia de color con la misma pieza entonces esto es un movimiento.
- Si hay dos casillas ocupadas y dos no ocupadas con el mismo color de pieza, entonces este es un movimiento especial llamado "enroque".
- Si hay una diferencia entre una casilla ocupada y una no ocupada con el mismo color de la pieza y una diferencia de una casilla no ocupada con el otro color de la pieza, entonces este es otro movimiento especial llamado "en passant".
- Si sólo hay una diferencia de casillas desocupadas y si hay un cambio de color de la pieza al color de la pieza anterior de la casilla desocupada en cualquier otra casilla ocupada, entonces este es un movimiento de captura
- Otro caso especial es la promoción de peón. En el juego de ajedrez, cuando un peón llega al otro extremo del tablero, al jugador se le ofrece promover el peón a cualquier pieza de su gusto. Si bien es común promover simplemente a una reina, uno debe tener la capacidad de elegir cualquier pieza. Así, una ventana de interfaz gráfica de usuario aparece pidiendo al usuario elegir a qué pieza le gustaría promover. Según el protocolo UCI, el carácter asociado con su elección se añade al final del movimiento. Por ejemplo, un movimiento “e7e8” con la elección de la promoción de una reina se convierte en “e7e8q”.

En la Tabla 4 , se puede observar un ejemplo de movimiento, en este caso, sería movimiento normal. El algoritmo detecta cuales son las casillas que tuvieron un mayor cambio de color, marca las mismas con dos cuadrados azules y envía la información al motor de ajedrez, que para la segunda fotografía sería “f2f4”, según la notación algebraica del ajedrez.

Tabla 4. Cambio de color detectado por el algoritmo de detección de movimiento

Estado Previo	Estado actual
	

4.2. Motor de ajedrez

En el ajedrez de computadora, un motor de ajedrez es un programa que analiza el ajedrez o las posiciones de las variantes de ajedrez, y genera una jugada o lista de jugadas que considera más fuerte. Un motor de ajedrez suele ser un back-end con una interfaz de línea de comandos sin gráficos ni ventanas. Los motores suelen utilizarse con un front-end, una interfaz gráfica con ventanas como Chessbase o WinBoard con la que el usuario puede interactuar a través de un teclado, un ratón o una pantalla táctil. Para este proyecto, el front-end servirá para que el usuario inicie una partida, indique si ha realizado un movimiento, seleccione dificultad entre otras. Este tema será abordado con mayor profundidad en el apartado de interfaz gráfica de usuario.

Para el correcto funcionamiento del motor de ajedrez, se utilizó dos herramientas: Stockfish, motor de ajedrez propiamente dicho y Python-chess, una biblioteca de Python con generación de movimientos, validación de movimientos y soporte para formatos comunes («python-chess: a pure Python chess library — python-chess 0.30.1 documentation», s. f.).

4.2.1. ¿Cómo funciona un motor de ajedrez tradicional?

Los motores de ajedrez son complejos. Sin embargo, en términos simples, hacen dos cosas importantes:

1. **Evaluar.** Los motores de ajedrez miran las posiciones individuales y evalúan qué posición es mejor. Casi todos los motores de ajedrez muestran un número de evaluación, o "eval", basado en la misma puntuación que la mayoría de los jugadores de ajedrez (un peón vale un punto, una pieza menor tres, etc.). Cada motor de ajedrez hace esto de manera diferente, pero la mayoría de los motores miran cosas como el material de cada lado, todas las amenazas en el tablero, la seguridad del rey y la estructura del peón.

   La puntuación acumulada de la mejor evaluación en el futuro se resume en un número. Los motores tradicionales se evalúan de forma similar a los humanos porque fueron diseñados por humanos. Los motores de red neuronal evalúan de forma diferente.

   La posición de abajo recibe una puntuación acumulativa de +3 por el motor informático Stockfish, aunque el material es igual, porque el desarrollo de las piezas de las Blancas es mucho mejor. Esto significa que la posición de las Blancas es aproximadamente tres peones mejor.

Figura 26. Posiciones de las piezas blancas y negras

2. **Búsqueda.** Como los buenos jugadores de ajedrez, los motores tratan de buscar profundamente en la posición. Cuanto más lejos puedan ver, mejor será la jugada que pueden hacer ahora, ya que pueden evaluar las posiciones que resultarán después de las mejores jugadas posibles en el futuro. Cada movimiento individual de ajedrez se llama "capa", y la profundidad se explica en cuántas capas de profundidad. A 20 capas (10 movimientos de las blancas y 10 de las negras), la mayoría de los motores ya están evaluando mucho más profundo y fuerte que los humanos. Dependiendo del tiempo permitido y de la complejidad de la posición, los motores pueden buscar en más de 50 capas de profundidad.

Desde la posición actual, un motor comienza a mirar todos los posibles movimientos y respuestas. Y luego todas las posibles respuestas a eso. ¡Y luego todas las posibles respuestas a eso! Imagina que hay 32 posibles movimientos en cualquier posición. Después de cuatro movimientos, ya hay más de un millón de posiciones para evaluar. Después de sólo cuatro movimientos más, eso sería más de un billón de posiciones. Eso se vuelve extremadamente poco práctico.

Así que, en lugar de eso, los motores tratan de usar "poda" inteligente para mirar profundamente sólo las líneas más prometedoras, e ignorar las obviamente malas. El motor mantiene en funcionamiento una "variación principal" (VP) de los movimientos más prometedores en cada posición.

Los motores de ajedrez tradicionales utilizan complejas funciones de evaluación y algoritmos de búsqueda inteligentes para encontrar la mejor jugada posible. Su potencia también está relacionada con la cantidad de potencia de procesamiento de la CPU que tiene el teléfono, el ordenador o el servidor. Cuanto más potentes y abundantes son las CPU, más fuerte se vuelve el motor.

4.2.2. ¿Qué es un motor de red neuronal, y en qué se diferencia?

Un motor de red neuronal (RN) es un tipo diferente de motor de ajedrez. El primer RN fue AlphaZero, creado por DeepMind (una compañía de Google). En 2017, AlphaZero supuestamente aplastó a Stockfish, el mejor motor tradicional, en una partida de 100 juegos. Pero la partida fue privada y muchos han cuestionado los resultados. Sin embargo, en 2019, el código abierto Lc0 (Leela Chess Zero), se convirtió finalmente en el motor de ajedrez más fuerte del mundo en el Campeonato de Ajedrez Informático de Chess.com.

Una red neuronal es una serie de algoritmos e instrucciones que se utilizan para evaluar la posición en el ajedrez, una RN se "entrena" alimentándola con datos (en este caso, juegos de ajedrez), y luego dejándola aprender por su cuenta. Esto es tradicionalmente llamado *machine learning*.

Los juegos pueden provenir de fuentes externas (como los juegos de gran maestro). O, como en Lc0, los datos del juego vienen de jugar más de 200 millones de partidas contra sí mismo. Así que, para los motores RN, su evaluación es proporcionada por la red neural.

La introducción de los motores RN también ha cambiado la forma en que se hace la búsqueda. Los motores tradicionales han usado típicamente lo que se llama una búsqueda minimax alfa-beta (AB), donde sólo se evalúan las mejores jugadas posibles. Los motores RN, sin embargo, eligen usar lo que se llama búsqueda de árbol de Monte Carlo (MCTS), donde se selecciona la mejor jugada en base a los resultados probables de muchos juegos. Básicamente, juega una tonelada de partidas rápidas contra sí mismo a velocidades súper rápidas con movimientos aleatorios y mira los movimientos que parecen tener las mayores probabilidades de ganar.

Los motores RN también se hacen más fuertes en base al tipo de hardware con el que se ejecutan. Necesitan CPUs potentes. Pero, aún más, necesitan GPUs potentes (unidades de procesamiento gráfico, como en muchos ordenadores de juegos), porque las GPUs son más rápidas en el procesamiento de las redes neuronales («Computer Chess Engines: A Quick Guide - Chess.com», s. f.).

Tabla 5. Top 10 de los mejores motores de ajedrez («CCRL 40/15 - Index», s. f.)

Ranking	Nombre	Elo	Oponente promedio	Partidas
1	Stockfish 11 64bi 4CPU	3494	-142.4	664
2	Lc0 0.22.0 T40B.4-160 GTX 1050	3463	-76.0	642
3	Komodo 13.3 64-bit 4CPU	3421	-66.8	442
4	Houdini 6 64-bit 4CPU	3399	-71.1	2100
5	Komodo 13.2.5 MC TS 64-bit 4CPU	3365	-12.0	524
6	Ethereal 11.75 64-bit 4CPU	3350	+15.0	614
7	Xiphos 0.6 64-bit 4CPU	3339	+37.1	638
8	Fire 7.1 64-bit 4CPU	3322	-11.1	1925
9	Laser 1.7 64-bit 4CPU	3285	+42.8	1073
10-11	Defenchess 2.2 64-bit 4CPU	3283	+57.3	828
10-11	RofChade 2.202 64-bit 4CPU	3283	+59.1	800

4.2.3. Puesta en marcha del motor de ajedrez

Para la creación de este módulo se empieza creando una clase llamada “ChessEng”, la cual importa la biblioteca Python-chess. Después, la clase inicializa un motor de ajedrez local utilizando el software de código abierto “Stockfish”. En la Figura 27, se puede ver que se inicial el motor de ajedrez, el protocolo UCI y se imprime el tablero de ajedrez en formato ASCII, un ejemplo del mismo se tiene en la Figura 28.

```
self.engBoard = chess.Board()
self.engine = chess.engine.SimpleEngine.popen_uci("./stockfish/Windows/stockfish_10_x64")
print(self.engBoard)
```

Figura 27. Constructor de la clase "ChessEng"

```
>>> board = chess.Board("r1bqkb1r/pppp1Qpp/2n2n2/4p3/2B1P3/8/PPPP1PPP/RNB1K1NR b KQkq - 0 4")
>>> print(board)
r . b q k b . r
p p p p . Q p p
. . n . . n . .
. . . . p . . .
. . B . P . . .
. . . . . . . .
P P P P . P P P
R N B . K . N R
```

Figura 28. Tablero en formato ASCII

Segundo, la biblioteca de Python-chess reconoce el movimiento y lo transforma a formato UCI, UCI es un sistema de comunicación abierto y transparente entre las partes del programa encargadas de realizar los cálculos ajedrecísticos y aquellas otras cuya tarea es facilitar la comunicación con el usuario, para después enviarlo al motor de ajedrez. Seguido de esto, se comprueba la legalidad del movimiento, para después actualizar las posiciones de las piezas en el tablero. La función que realiza todo este procedimiento se encuentra descrita en la

```python
def updateMove(self, move):
    '''
    Actualiza el tablero de ajedrez con el movimiento realizado.
    También verifica movimientos ilegales
    '''

    # Convierte el movimiento a formato UCI para el motor de ajedrez
    uciMove = chess.Move.from_uci(move)

    # Comprueba la legalidad
    if uciMove not in self.engBoard.legal_moves:
        return 1
    else:
        # Actualiza el tablero
        self.engBoard.push(uciMove)
        print(self.engBoard)
        return 0
```

Figura 29. Método "updateMove"

Por último, se obtiene el mejor movimiento del motor de ajedrez Stockfish, que será transmitido por medio de un socket al robot. El método funciona de la siguiente manera: se da la posición actual del tablero, para que el motor de ajedrez analice la mejor jugada. Se verifica si es algún tipo de movimiento es, por ejemplo: ataque o enroque. Por último, se devuelve la mejor jugada y se actualiza el tablero. Este método se lo puede observar en la Figura 30.

```
def feedToAI(self):
    '''
    Obtiene el mejor movimiento del motor stockfish.
    Escribe el movimiento elegido al archivo Game.txt
    '''

    # Da al CPU la posición actual del tablero
    response = self.engine.play(self.engBoard, chess.engine.Limit(time=0.200))

    # Da información sobre qué tipo de movimiento es, ataque o enroque.

    attack = str(self.engBoard.is_capture(response.move))
    castling = str(self.engBoard.is_castling(response.move))
```

Figura 30. Método "feedToAI"

4.3. Interfaz gráfica de usuario

En este apartado, se creará una interfaz gráfica de usuario (GUI), por sus siglas en inglés. Esta interface se utilizará para que el oponente humano pueda interactuar con el software del aplicativo o back-end del mismo.

Para la creación del GUI, lo que haremos será crear botones, los que servirán para navegar entre marcos y ventanas. Lo que se utilizará aquí será una metodología para añadir páginas básicamente hasta el infinito.

```
class Application(tk.Tk):
    '''
    Esta clase controla la Interfaz Gráfica de Usuario (GUI)
    '''
    def __init__(self,*args, **kwargs):

        tk.Tk.__init__(self,*args,**kwargs)

        container = tk.Frame(self)

        container.pack(side = "top", fill = "both", expand = True)

        container.grid_rowconfigure(0,weight = 1)
        container.grid_columnconfigure(0, weight = 1)

        self.frames = {}
        self.game = Game()

        # Contiene la información de movimiento de la CPU que se mostrará en CPUMovePage
        self.move = StringVar()
        self.move.set("e2")

        # Contiene la información del ganador que se mostrará en GameOverPage
        self.winner = StringVar()
        self.winner.set("¡CPU gana!")

        # Da objetos de página a la aplicación para mostrar en el frame
        for F in (StartGamePage, InitializeBoardPage,SetBoardPage, ChooseColorPage,
                ChooseDifficultyPage, CPUMovePage, PlayerMovePage, CheckPage,
                CPUMoveErrorPage, GameOverPage, PlayerMoveErrorPage, ChoosePromotionPage):

            frame = F(container, self)
```

```
            self.frames[F] = frame
            frame.grid(row = 0, column = 0, sticky = "nsew")

        self.show_frame(StartGamePage)


    def show_frame(self,cont):
        '''
        Eleva el marco a la parte superior, mostrándolo como la ventana actual
        '''

        frame = self.frames[cont]
        frame.tkraise()
```

Figura 31. Creación de la ventana que albergará los frames

Con el método *show_frame()* de la Figura 31, lo que hacemos es llenar esta tupla con todas las páginas posibles de nuestra aplicación. Esto cargará todas estas páginas para nosotros. Dentro de nuestro método __init__, llamamos primero a StartGamePage para mostrar, aunque después podemos llamar a show_frame para elevar cualquier otro frame/ventana que queramos.

Así que hemos creado InitializeBoardPage, SetBoardPage, ChooseColorPage, ChooseDifficultyPage, CPUMovePage, PlayerMovePage, CheckPage, CPUMoveErrorPage, GameOverPage, PlayerMoveErrorPage, ChoosePromotionPage. Necesitamos tener algo de que nos permita navegar hacia estas páginas desde StarGamePage. De esta manera, queda nuestra StarGamePage:

```
class StartGamePage(tk.Frame):
    '''
    Solicita al usuario que comience un juego nuevo
    '''

    def __init__(self,parent,controller):

        tk.Frame.__init__(self,parent)
        controller.title("Robot autónomo para juego de ajedrez")
        controller.iconbitmap('logo_UPC.ico')
        controller.resizable(width=False, height=False)
        # Coloca una etiqueta
        label = tk.Label(self,text="Robot Ajedrecista, usando un sistema de visión artificial", font=LARGE_FONT)
        label.pack(pady=20, padx=20)

        # Crea el botón que te lleva a la página de inicialización del tablero y llama a Game.setUp()
        startGameButton = tk.Button(self, text="Comenzar Juego Nuevo",font=MED_FONT,
                        command=lambda: [controller.show_frame(InitializeBoardPage), controller.game.setUp()])
        startGameButton.pack()
```

Figura 32. Frame "StartGamePage"

Añadir botones de esta manera en todas las páginas que se irán mostrando, permitirá al usuario navegar a través del aplicativo. Esta interfaz gráfica de usuario tendrá un diagrama de flujo determinador, el mismo que se detalla en la Figura 33.

En el flujo de trabajo, se puede observar que si el jugador elige las piezas verdes (blancas), moverá primero. Si el jugador elige rojas (negras), el robot mueve primero. Este orden de jugadores viene dictado por las normas del ajedrez.

El código comprobará si hay algún error en cada movimiento. Si hay un error, aparecerá un error en la interfaz gráfica de usuario, evitando que el jugador pase a la siguiente fase, y le pedirá que haga un movimiento válido. El flujo de movimiento de jugador a movimiento de la CPU y viceversa continuará cuando el jugador haga un movimiento válido.

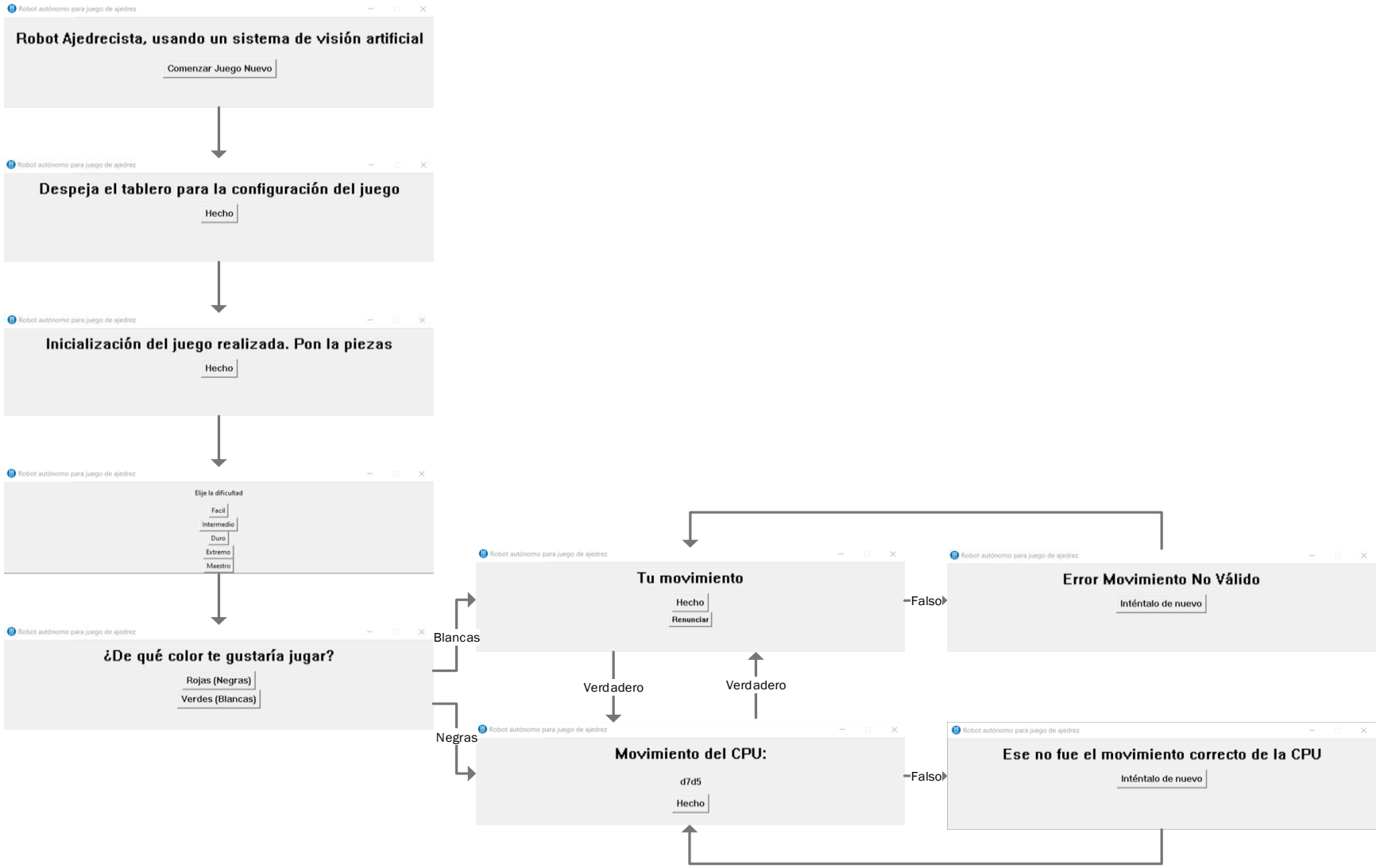


Figura 33. Flujo de trabajo interfaz gráfica de usuario

4.4. Interfaz de comunicación

En este apartado se describe la creación de una interfaz de comunicación entre el aplicativo y el robot. Para este caso, se utilizará una metodología cliente-servidor, en el que el cliente será el aplicativo desarrollado en Python, y el servidor será la controladora IRC5, del robot. El método de conexión que se usó es el socket, que sirve principalmente para conectar dos programas que se encuentran corriendo en la red. Todo el proceso de creación de la interfaz se explica a continuación.

4.4.1. Socket

Un socket es un punto final interno para enviar o recibir datos dentro de un nodo de una red informática. El término socket es análogo a los conectores físicos hembra, la comunicación entre dos nodos a través de un canal se visualiza como un cable con dos conectores macho que se conectan a los sockets de cada nodo. Análogamente, el término puerto (otro término para un conector hembra) se utiliza para los puntos finales externos de un nodo, y el término zócalo también se utiliza para un punto final interno de la comunicación entre procesos locales (IPC) (no a través de una red). Sin embargo, la analogía es limitada, ya que la comunicación de red no tiene por qué ser individual o tener un canal de comunicación dedicado.

4.4.1.1. Funciones de la API del socket

El API del socket de Berkeley típicamente provee las siguientes funciones:

- *socket()* crea un nuevo socket de un cierto tipo, identificado por un número entero, y le asigna recursos del sistema.
- *bind()* se utiliza típicamente en el lado del servidor, y asocia un socket con una estructura de direcciones de socket, es decir, una dirección IP local específica y un número de puerto.
- *listen()* se utiliza en el lado del servidor, y hace que un socket TCP vinculado entre en estado de escucha.
- *connect()* se utiliza en el lado del cliente, y asigna un número de puerto local libre a un socket. En el caso de un socket TCP, causa un intento de establecer una nueva conexión TCP.
- *accept()* se utiliza en el lado del servidor. Acepta un intento entrante recibido de crear una nueva conexión TCP desde el cliente remoto, y crea un nuevo socket asociado al par de direcciones de socket de esta conexión.
- *send(), recv(), sendto(), y recvfrom()* se utilizan para enviar y recibir datos. También se pueden utilizar las funciones estándar write() y read().
- *close()* hace que el sistema libere los recursos asignados a un socket. En el caso del TCP, la conexión se termina.
- *gethostbyname() y gethostbyaddr()* se utilizan para resolver nombres y direcciones de host. Sólo IPv4.
- *select()* se utiliza para suspender, esperando que uno o más de una lista proporcionada de sockets estén listos para leer, listos para escribir, o que tengan errores.
- *poll()* se utiliza para comprobar el estado de un socket en un conjunto de sockets. El conjunto puede ser probado para ver si se puede escribir en algún socket, leer de él o si se ha producido un error.
- *getsockopt()* se utiliza para recuperar el valor actual de una opción de un determinado socket para el socket especificado.
- *setsockopt()* se utiliza para establecer una opción de socket particular para el socket especificado.

En el siguiente diagrama, veamos la secuencia de llamadas a la API de los sockets y el flujo de datos para el TCP:

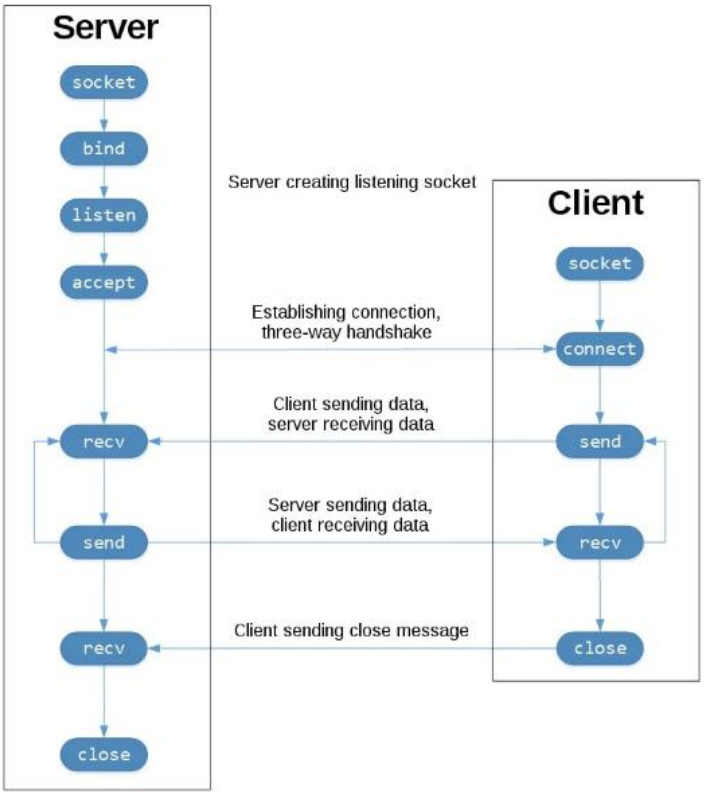


Figura 34. Comunicación cliente-servidor por sockets

Para la creación de la interfaz de comunicación se creará el cliente en Python y el servidor en RobotStudio.

Primero, se creará el cliente. Los pasos necesarios para la creación del cliente son los siguientes:

- Primero, importar la biblioteca del módulo de socket.
- Segundo, se debe crear un objeto de socket.
- Después, se define un puerto en el cual se encuentra nuestro localhost (puerto en el cual el servidor corre).
- Por último, se envía y recibe la información desde y hacia el servidor. En este caso la información enviada será las casillas desde donde parte la pieza hasta la que llega. También se envía que tipo de movimiento, es decir, ataque y enroque. Adicionalmente, se recibe la respuesta del servidor y se cierra la conexión.

El código necesario para la creación del socket cliente se lo puede ver en la Figura 35.

```python
# Importar el módulo del socket
import socket

# Crea un objeto socket
s = socket.socket()
# Define el puerto al cuál se quiere conectar
port = 55000
# conecta al servidor local de la computadora
s.connect(('127.0.0.1', port))
# recibe y envía datos desde y hacia el servidor
while True:
    if i == 0:
        s.send(bytes(a, 'utf-8'))
        i = i + 1
        s.recv(4096)
    elif i == 1:
        s.send(bytes(b, 'utf-8'))
        i = i + 1
        s.recv(4096)
    elif i == 2:
        s.send(bytes(attack, 'utf-8'))
        i = i + 1
        s.recv(4096)
    elif i == 3:
        s.send(bytes(castling, 'utf-8'))
        i = i + 1
        s.recv(4096)
    elif i > 3:
        break
# cierra la conexión
s.close()
```

Figura 35. Creación del socket cliente

Segundo, se procederá a crear el servidor, esto se realizará en RobotStudio, pero para realizar esto se debe configurar unos parámetros en el programa para que la comunicación del socket funcione, que son los siguientes:

- Se selecciona un robot de la biblioteca de ABB, que para nuestro caso será el IRB 140, Figura 36.

Figura 36. Selección del robot IRB 140

- Después se le añade un sistema de robot, que será la controladora virtual del mismo, Figura 37. Se da clic en siguiente hasta que aparezca la ventana de opciones del sistema, Figura 38.

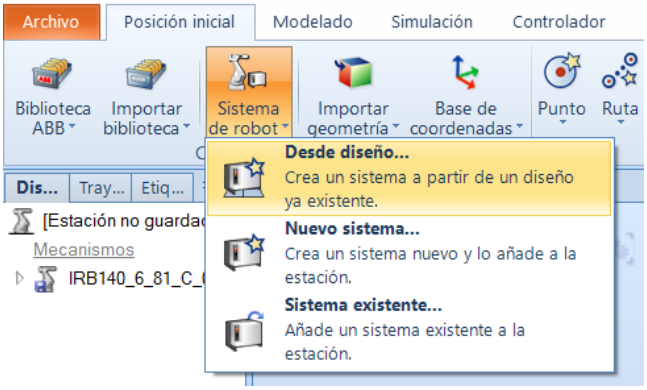


Figura 37. Creación de un sistema a partir de un diseño ya existente

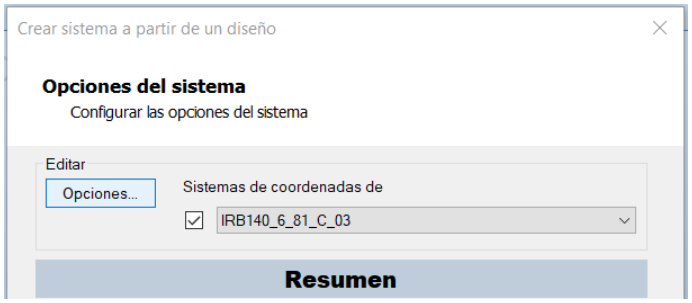


Figura 38. Opciones del sistema

- Una vez abierta la ventana de opciones se procede a dar clic en "*Communication*" y después se selecciona la opción "616-1 PC Interface", Figura 39.

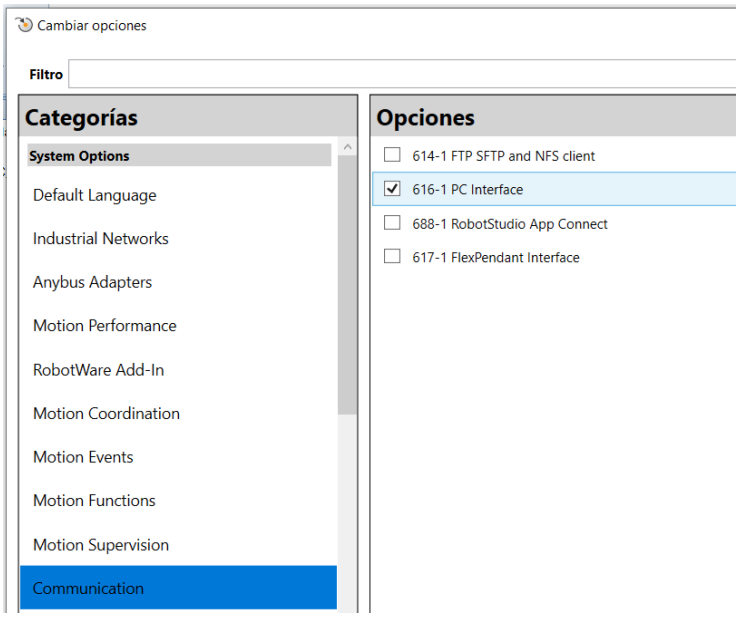


Figura 39. Opciones de sistema, comunicaciones

Finalizado estos pasos, el programa estará listo para crear la comunicación por sockets que estará escrito en lenguaje *RAPID*, que es propio de RobotStudio. Las instrucciones que se utilizarán para este caso serán las siguientes:

- *socketdev* se usa para comunicarse con otros ordenadores en una red o entre tareas de RAPID.
- *SocketCreate* se usa para crear un nuevo zócalo para una comunicación basada en conexiones. El intercambio de mensajes de zócalo es de tipo canal y tiene el protocolo TCP/IP con garantía de entrega. Es posible desarrollar aplicaciones tanto de servidor como de cliente.
- *SocketBind* se usa para enlazar un zócalo al número de puerto de servidor y la dirección IP y el número de puerto especificados. Este solo puede usarse con aplicaciones de servidor.
- *SocketListen* se usa para iniciar la escucha de conexiones entrantes, por ejemplo, para empezar a actuar como servidor. Sólo se puede usar en aplicaciones de servidor.
- *SocketAccept* se utiliza para aceptar peticiones de conexión entrantes. Solo puede usarse con aplicaciones de servidor.
- *SocketReceive* se usa para recibir datos de un ordenador remoto. Puede usarse tanto en aplicaciones de cliente como en aplicaciones de servidor.
- *SocketSend* se usa para enviar datos a un ordenador remoto. Puede usarse tanto en aplicaciones de cliente como en aplicaciones de servidor.
- *SocketClose* se utiliza cuando ya no se va a utilizar una conexión de zócalo. Una vez cerrado un zócalo, no es posible utilizarlo en ninguna llamada a zócalo, excepto *SocketCreate* (ABB, 2004).

Ahora se escribirá el código en RAPID, se comenzará con la declaración de variables como se observa en la Figura 40 .

```
VAR socketdev server;
VAR socketdev client;
VAR string message;
VAR num coordenada;
VAR string receiveString;
VAR string PuntoOrigen;
VAR string PuntoDestino;
VAR string Ataque;
VAR string Enroque;
```

Figura 40. Declaración de variables

Después se crea la comunicación del socket en el que se ingresa la IP que viene por defecto en la controladora virtual de RobotStudio que es: “127.0.0.1”, esta IP y el puerto serán los mismos en el cliente para que se pueda establecer la comunicación. Cuando se trabaje con el robot real se cambiará la dirección por la que tiene el mismo. La Figura 41, muestra la creación de la comunicación por sockets.

```
PROC createServer()

        ! Crea la comunicación
        SocketCreate server;
        SocketBind server,"127.0.0.1",55000;
        SocketListen server;
        SocketAccept server,client\Time:=WAIT_MAX;

ENDPROC
```

Figura 41. Creación de la comunicación en el servidor

Finalmente, se crea el procedimiento que servirá para recibir y enviar datos al cliente. En este caso se recibirá las casillas de partida y destino de las piezas, así como el tipo de movimiento que se realizó. El servidor por su parte enviará una respuesta, informando que la información ha sido recibida correctamente. En la *Figura 42*, se observa en detalle el código empleado.

```
PROC ReceiveInformation()
        FOR count FROM 0 TO 3 DO
           IF count = 0 THEN
            SocketReceive client,\Str:=receiveString\Time:=WAIT_MAX;
            PuntoOrigen:=receiveString;
            SocketSend client,\Str:="Punto de origen";
            ELSEIF count = 1 THEN
            SocketReceive client,\Str:=receiveString;
            PuntoDestino:=receiveString;
            SocketSend client,\Str:="Punto de destino";
            ELSEIF count = 2 THEN
            SocketReceive client,\Str:=receiveString;
            Ataque:=receiveString;
            SocketSend client,\Str:="Ataque";
            ELSEIF count = 3 THEN
            SocketReceive client,\Str:=receiveString;
            Enroque:=receiveString;
            SocketSend client,\Str:="Enroque";
            ENDIF
        ENDFOR
    count:=0;
ERROR
    IF ERRNO=ERR_SOCK_TIMEOUT THEN
        RETRY;
    ELSEIF ERRNO=ERR_SOCK_CLOSED THEN
        RETURN ;
    ELSE
        ! Sin manejo de recuperación de errores
    ENDIF

    ! Cerrar comunicación
    SocketClose server;
ENDPROC
```

Figura 42. Código para recibir y enviar información en el servidor

4.5. Software de la unidad central de control

El software de la unidad central de control, es un sistema que se utiliza como puente principal entre la interfaz gráfica de usuario, procesamiento de imágenes, motor de ajedrez e interfaz de comunicación entre el aplicativo y el robot. Este software va a estar contenido en una clase que se llamará *"Game"*. Esta clase llama a la inicialización del tablero, que servirá para guardar el estado inicial del mismo y el estado en el que se encuentra el juego. Además, alberga los métodos: *playerMove, playerPromotion, CPUMove, updateCurrent* y *setUp*. Todos estos métodos se explicarán a continuación.

4.5.1. __init__()

Este es un método especial de una clase en Python. El objetivo fundamental es crear un constructor, mismo que servirá para inicializar los valores iniciales de un nuevo objeto, en este caso, inicializa el objeto Game y crea varias variables booleanas con respecto al estado del juego. También establece el ganador del juego como marcador de posición. En la Figura 43, se encuentran que variables se crearon.

```
def __init__(self):

    self.over = False
    self.CPUMoveError = False
    self.PlayerMoveError = False
    self.isCheck = False
    self.winner = "YO"
```

Figura 43. Constructor de la clase "Game"

4.5.2. setUp()

Este método se usará para inicializar los objetos con los que interactuará el juego, como son: la cámara (toma de fotografías del tablero y sus piezas), el motor de ajedrez (encargado del análisis de movimientos y jugadas de ajedrez), el socket (interfaz de comunicación del cliente), el tablero, las posiciones de las piezas actual y previa, y también el último movimiento del CPU.

```
def setUp(self):

    self.camera = Camera()
    self.chessEngine = ChessEng()
    self.zocalo = Socket()
    self.board = 0
    self.current = 0
    self.previous = 0
    self.CPULastMove = "0"
```

Figura 44. Método setUp()

4.5.3. analyzeBoard()

Método que llama a "*board_Recognition*" para tomar una imagen del tablero, analizarla (aquí se utiliza el algoritmo de reconocimiento de imágenes, que fue explicado en secciones anteriores),

después se inicializa el tablero y se le asigna un estado inicial (se crea una matriz del tablero de ajedrez con todas las piezas y configuraciones primeras).

```
def analyzeBoard(self):

    boardRec = board_Recognition(self.camera)
    self.board = boardRec.initialize_Board()
    self.board.assignState()
```

Figura 45. Método analyzeBoard()

4.5.4. checkBoardIsSet()

En este método se utiliza una sola vez en cada partida, dado que, toma una fotografía inicial del tablero armado. También actualiza el estado actual del tablero para de esta manera comenzar el juego.

```
def checkBoardIsSet(self):

    self.current = self.camera.get_frames()
```

4.5.5. playerMove()

Método que sirve para comparar el estado o las posiciones anteriores de las piezas en el tablero de ajedrez con las actuales, para determinar el movimiento realizado por el jugar. El movimiento realizado se envía al motor de ajedrez para actualizar la posición del tablero y que se devuelva una jugada apropiada. También se revisa si el movimiento ejecutado por el jugador humano es legal o ilegal y finalmente también se revisa si el juego ha terminado.

```
def playerMove(self):

    self.previous = self.current
    self.current = self.camera.get_frames()
    move = self.board.determineChanges(self.previous,self.current)
    code = self.chessEngine.updateMove(move)
    if code == 1:
        # Movimiento ilegal despliega una página de error de movimiento del jugador
        self.PlayerMoveError = True
    else:
        self.PlayerMoveError = False
        # Escribe al archivo Game.txt
        f = open("Game.txt", "a+")
        f.write(chess.Move.from_uci(move).uci() + "\r\n")
        f.close()

    # Comprueba si se acabo el juego
    if self.chessEngine.engBoard.is_checkmate():
        self.winner = "¡Tú ganas!"
        self.over = True
```

Figura 46. Método playerMove()

4.5.6. playerPromotion()

Este método es usado, así como el anterior, para comparar dos estados del tablero de ajedrez, uno anterior y el otro actual. La diferencia con el método playerMove(), es que este método sirve para identificar cuando se ha realizado un movimiento de coronación de peón, al cual se lo cambiará por otra pieza, generalmente una dama.

```
def playerPromotion(self, move):

    print(move)
    code = self.chessEngine.updateMove(move)
    if code == 1:
        # Movimiento ilegal despliega una página de error de movimiento del jugador
        print("Error")
        self.PlayerMoveError = True
    else:
        self.PlayerMoveError = False

        # Escribe al archivo Game.txt
        f = open("Game.txt", "a+")
        f.write(chess.Move.from_uci(move).uci() + "\r\n")
        f.close()

    # Comprueba si se acabo el juego
    if self.chessEngine.engBoard.is_checkmate():
        self.winner = "¡Tú ganas!"
        self.over = True
```

Figura 47. Método playerPromotion()

4.5.7. CPUMove()

Método que obtiene el movimiento del CPU, después de que el jugador ha realizado su movimiento o cuando el CPU juega con blancas en el primer turno, que viene desde el motor de ajedrez. En este caso da dos avisos: el primero es si el jugador se encuentra en jaque, el programa alertará de ello y segundo si el juego se ha terminado, es decir, si la computadora ha realizado jaque mate.

```
def CPUMove(self):

    # Obtiene el movimiento desde el motor de ajedrez
    self.CPULastMove = self.chessEngine.feedToAI()

    # Si está en jaque el GUI abrirá una ventana alertando al usuario del Jaque
    self.isCheck = self.chessEngine.engBoard.is_check()

    # Comprueba si se acabo el juego
    if self.chessEngine.engBoard.is_checkmate():
        self.winner = "¡CPU gana!"
        self.over = True

    return self.CPULastMove
```

Figura 48. Método CPUMove

4.5.8. updateCurrent()

Al igual que los métodos: playerMove y playerPromotion, este método compara la imagen anterior del tablero con la imagen actual para actualizarla. Con este método se asegura que el robot haya movido la pieza de manera correcta. En este caso se desplegará una página de error cuando el movimiento ejecutado no es el adecuado.

```
def updateCurrent(self):

    self.previous = self.current
    self.current = self.camera.get_frames()

    # determina el movimiento
    move = self.board.determineChanges(self.previous, self.current)
    move = chess.Move.from_uci(move)

    # Asegura que el jugador haya movido la pieza de la CPU correctamente
    if move == self.CPULastMove:
        self.CPUMoveError = False
    else:
        # GUI abrirá una página de error de movimiento del CPU
        self.CPUMoveError = True
```

Figura 49. Método updateCurrent()

4.6. Programación de movimientos del robot ABB IRB 140

En esta sección se explica la programación de los movimientos que va a ejecutar el robot IRB 140 en una partida de ajedrez contra un jugador humano. Las instrucciones vendrán del aplicativo creado en

Python, mismas que serán interpretadas por la controladora del robot, la IRC5, para generar un movimiento. Los movimientos que realizará el robot serán: un movimiento regular (mover la pieza de una casilla a otra), movimiento de captura (cuando una pieza ataca a otra, el robot retira la pieza atacada y coloca la segunda en la posición de la primera), movimiento de enroque (el rey y la torre son movidos, el movimiento dependerá si es enroque corto o largo). Adicionalmente, el robot utilizará un elemento terminal que servirá para sujetar las piezas, este elemento terminal realizará cuatro procedimientos: Abrir pinza, sostener pieza, soltar pieza y cerrar pinza.

4.6.1. Conceptos básicos de RAPID

4.6.1.1. ¿Qué es RAPID?

Tradicionalmente los ordenadores poseen un lenguaje nativo de unos y ceros, un lenguaje que es de muy difícil comprensión para las personas. Por otro lado, RAPID es un lenguaje de programación de alto nivel, es decir, es un software que es comprensible para las personas, ya que, utiliza palabras en inglés (como IF Y FOR) para fácil escritura y lectura del programa. Este lenguaje se utiliza para que el robot realice tareas que nosotros le asignemos.

4.6.1.2. Datos de RAPID

4.6.1.2.1 Variables

El tipo de variables con los que cuenta RAPID son muy diversos, pero para este trabajo nos centraremos específicamente en cuatro tipos, de los que se puede ver en la Tabla 6.

Tabla 6. Tipos de datos RAPID (ABB, 2007)

Tipo de dato	**Descripción**
Num	Números, que pueden ser decimales o enteros, por ejemplo 5 o 3,14159
String	Una cadena de texto. Por ejemplo “Hola mundo”. Máximo 80 caracteres
Bool	Una variable booleana (lógica). Sólo puede tomar valores de TRUE o FALSE
Pers	Es igual a una variable normal, excepto porque recuerda el último valor que se le haya asignado.

4.6.1.2.2 Constantes

Los valores de las constantes son asignados en la declaración y no pueden ser modificados luego en alguna parte del programa. Para declarar una constante se utiliza la palabra clave CONST seguida del tipo de dato, el identificador y la asignación de un valor, por ejemplo:

```
CONTS  num gravedad := 9.81;

CONTS string saludo := “Hola”
```

4.6.1.2.3 Operadores

En RAPID existen tres tipos de operadores: numéricos, relacionales y de cadena. Estos operadores están descritos a detalle en las siguientes tablas:

Tabla 7. Operadores numéricos

Operador	Descripción	Ejemplo
+	Suma	Reg1 := reg2 + reg3;
-	Resta Menos unario	Reg1 := reg2 - reg3;
*	Multiplicación	Reg1 := reg2 * reg3;
/	División	Reg1 := reg2 / reg3;

Tabla 8. Operadores relacionales

Operador	Descripción	Ejemplo
=	Igual a	Flag1 : = reg1 = reg2; Flag1 es TRUE si reg1 es igual a reg2
<	Menor que	Flag1 : = reg1 < reg2; Flag1 es TRUE si reg1 es menor que reg2
>	Mayor que	Flag1 : = reg1 > reg2; Flag1 es TRUE si reg1 es mayor que reg2
<=	Menor o igual que	Flag1 : = reg1 <= reg2; Flag1 es TRUE si reg1 es menor que o igual a reg2
>=	Mayor o igual que	Flag1 : = reg1 >= reg2; Flag1 es TRUE si reg1 es mayor que o igual a reg2
< >	Distinto de	Flag1 : = reg1 < > reg2; Flag1 es TRUE si reg1 es distinto de reg2

Operador	Descripción	Ejemplo
+	Concatenación de cadenas	VAR string firstname := “John”; VAR string lastname := “Smith”; VAR string fullname; Fullname :=firstname + “ ” + Lastname; La variable fullname contendrá la cadena “John Smith”

4.6.1.3. Terminología de la estructura de RAPID

La Tabla 9 describe la terminología de RAPID con la que se trabajó con RobotStudio. Los conceptos aparecen enumerados por su dimensión, de los más básicos a los de mayor envergadura.

Tabla 9. Terminología de la estructura de RAPID (Robotics, 2019)

Concepto	**Explicación**
Declaración de datos	Se utilizan para crear instancias de variables o tipos de datos, como num o tooldata.
Instrucción	Los comandos de código en sí que hacen que ocurra algo, por ejemplo, el cambio de un dato a un valor determinado o un movimiento del robot. Las instrucciones sólo pueden ser creadas dentro de una rutina.
Instrucciones de movimiento	Crean los movimientos del robot. Se componen de una referencia a un objetivo, especificado en una declaración de datos, junto con parámetros que establecen el comportamiento de los movimientos y procesos. Si se usan objetivos en línea, la posición se declara en las instrucciones de movimiento.
Instrucción de acción	Instrucciones que hacen cosas distintas que mover el robot, por ejemplo, cambiar el valor de un dato o definir las propiedades de sincronización.

Rutina	Normalmente, un conjunto de declaraciones de datos, seguido de un conjunto de instrucciones utilizadas para implementar una tarea. Las rutinas pueden dividirse en tres categorías: procedimientos, funciones y rutinas TRAP.
Procedimiento	Un conjunto de instrucciones que no devuelve ningún valor.
Función	Un conjunto de instrucciones que devuelve un valor.
Rutina TRAP	Un conjunto de instrucciones que es disparado por una interrupción.
Módulo	Un conjunto de declaraciones de datos, seguido de un conjunto de rutinas. Los módulos pueden ser guardados, cargados y copiados en forma de archivos. Los módulos se dividen entre módulos de programa y módulos de sistema.
Módulo de programa (.mod)	Pueden ser cargados y descargados durante la ejecución.
Módulo de sistema (.sys)	Se utilizan principalmente para datos y rutinas específicos del sistema, por ejemplo, un módulo de sistema de ArcWare que es común para todos los robots de soldadura al arco.
Archivos de programa (.pgf)	En IRC5, un programa de RAPID es una colección de archivos de módulo (.mod) y el archivo de programa (.pgf.) que hace referencia a todos los archivos de módulo. Al cargar un archivo de programa, todos los módulos de programa antiguos se reemplazan por aquéllos a los que se hace referencia en el archivo .pgf. Los módulos de sistema no se ven afectados por la carga de programas.

4.6.2. Programación del robot

4.6.2.1. Flujo de trabajo de la programación del robot

En gran parte de los programas desarrollados, lo más recomendable es recorrer los mismo (programas) desde el principio hacia el final, a pesar de que es posible realizar este trabajo siguiendo otras secuencias. Como requisito previo a la programación se debe cumplir algunos parámetros como: configurar la estación, incluido el robot, los útiles y las piezas de trabajo.

En la Tabla 10 se describe el flujo que va a realizar el programa, para que el robot realice una tarea determinada.

Tabla 10. Flujo de trabajo de la programación del robot

Tarea	**Descripción**
Creación de objetivos y trayectorias	Se crea los objetivos y las trayectorias que son necesarias para que el robot realice el trabajo. Para crear los objetivos y trayectorias, se debe usar los métodos descritos a continuación. - Crear una curva que concuerde con la forma necesaria. Después,

	se utiliza el comando Crear trayectoria a partir de curva para generar una trayectoria, con sus objetivos, a lo largo de la forma creada. - Crear los objetivos en las posiciones necesarias, para después crear una trayectoria insertando en ella los objetivos creados.
Comprobación de las orientaciones de los objetivos	Cerciorarse de que los objetivos estén alineados de la forma más eficaz para las tareas a ejecutar. Caso contrario, reoriente los objetivos hasta que esté conforme.
Comprobación de la alcanzabilidad	Verificar que el robot y la herramienta puedan conseguir todos los objetivos de la trayectoria.
Sincronización del programa con el controlador virtual	Crea código de RAPID desde los elementos de RobotStudio y habilita la simulación del programa.
Realización de la edición basada en texto	Si se necesita modificar las instrucciones o los datos elaborados mediante RobotStudio, se puede ejecutar el Editor de RAPID.
Detección de colisiones	Comprobar que el robot no colisione con ningún equipo circundante o con algún tipo de útil. Si se produce ello, corregir las posiciones hasta que no exista ningún tipo de colisión.
Prueba del programa	Probar el programa moviéndose a lo largo de todas las trayectorias creadas.

4.6.2.2. Lógica de programación

La principal finalidad de este programa que pondrá en funcionamiento al robot es: leer los datos externos que vienen del aplicativo de visión por computadora y motor de ajedrez, el mismo que enviará los movimientos, en este caso las casillas de origen y destino según el tipo de movimiento que se haya realizado. Para que el programa en RobotStudio y el aplicativo se comuniquen se utilizó sockets, estos fueron explicados en la sección de interfaz de comunicación.

La creación del servidor en RobotStudio fue explicada en el apartado de interfaz de comunicación, por lo que se omitirá la explicación del mismo en esta sección. Se explicará el flujo de programa que ejecutará los movimientos. Este flujo puede ser apreciado de mejor manera en la Figura 51.

4.6.2.2.1 Puntos de referencia fijos

Para comenzar con el programa en RobotStudio se comenzará creando los puntos más importantes que con los que el robot trabajará. Estos puntos son: *Home* (es la posición en la que el robot se encuentra en "descanso"), *StartingReference* (coordenadas del centro de la primera casilla o escaque del tablero de ajedrez), *Reference1* (coordenadas de la esquina derecha del tablero más cercana al robot), *Reference2* (coordenadas de la esquina izquierda del tablero más lejana al robot). Las dos últimas referencias será utilizadas para determinar si el robot puede alcanzar esos puntos, ya que son posiciones extremas. En la Figura 50, se puede apreciar la sintaxis que se utilizó para determinar estos puntos.

```
MODULE Module1
    CONST robtarget Home:=[[689.268487273,0,537.14730631],[0.190808996,0,0.981627183,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget StartingReference:=[[425,-175,210],[0,0,1,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Reference2:=[[800,200,210],[0,0,1,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Reference1:=[[400,-200,210],[0,0,1,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST num safe_height := 110;
    CONST robtarget Discard:=[[500,-400,250],[0.190808995,0,0.981627183,0],[-1,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

Figura 50 Principales puntos de referencia

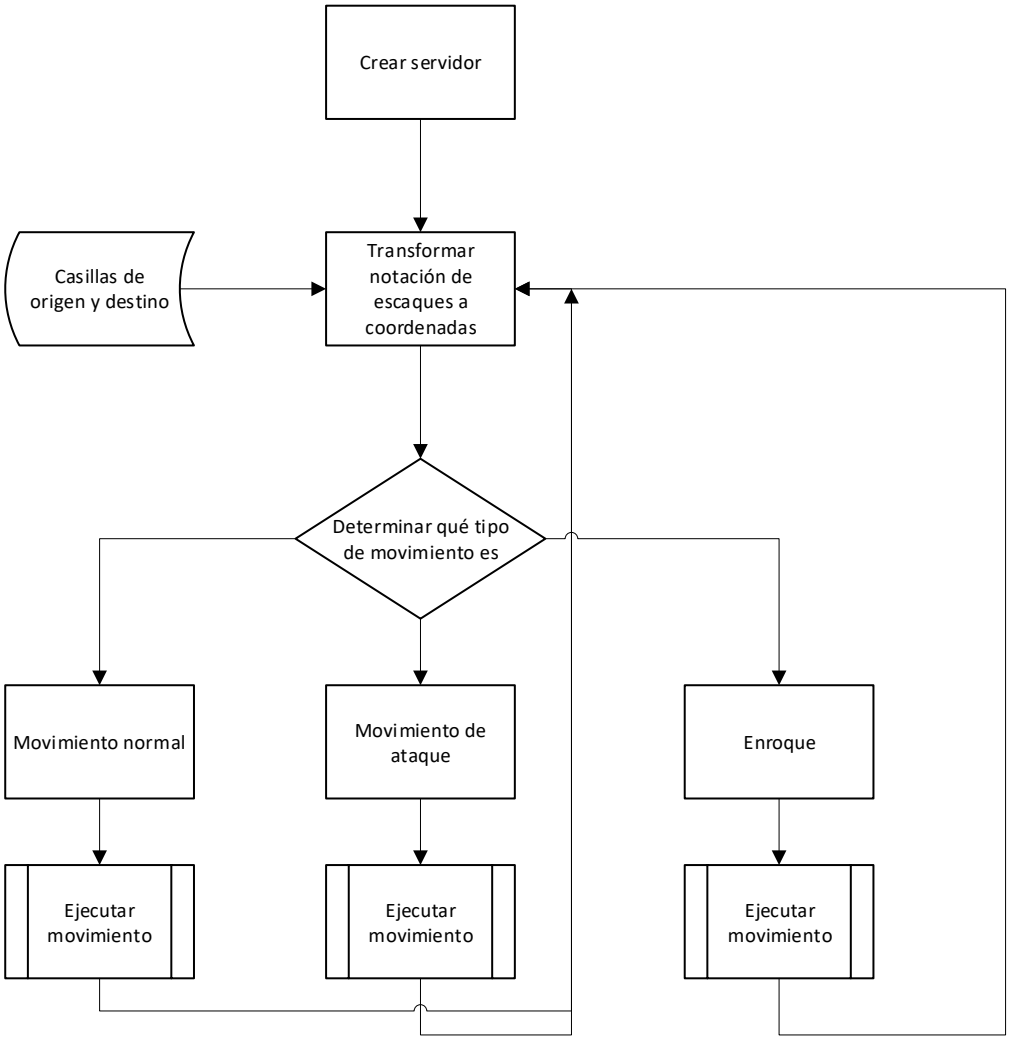


Figura 51. Flujo de trabajo programa en RAPID

4.6.2.2.2 Variables del programa

Los puntos de referencia anteriormente descritos, son puntos fijos, es decir, permanecerán contantes en todo el flujo de trabajo del robot. Por otra parte, el programa necesita puntos variables, mismos que irán cambiando a lo largo de la ejecución del programa. Estos objetivos variables irán tomando valores según los escaques que determine el aplicativo de Python. En la Figura 52 se observa las variables creadas para este propósito. Tanto las variables “a” como “b”, serán las coordenadas de las casillas de origen y las variables “x” e “y”, serán las coordenadas de las casillas de destino.

```
VAR num a;
VAR num b;
VAR num x;
VAR num y;
```

Figura 52. Puntos variables de trabajo del robot

Las variables de la *Figura 52* se transformarán en coordenadas dependiendo que valor venga del aplicativo de visión por computadora. Por ejemplo, una jugada muy habitual es la de apertura de peón de rey, *Figura 53*, que transformada a notación algebraica de ajedrez será *"e2e4"*. El movimiento *"e2e4"* indica dos casillas, una de origen y otra de destino, en el programa de RAPID para este caso las coordenadas serán transformadas como se ve en: *Figura 54* y *Figura 55*

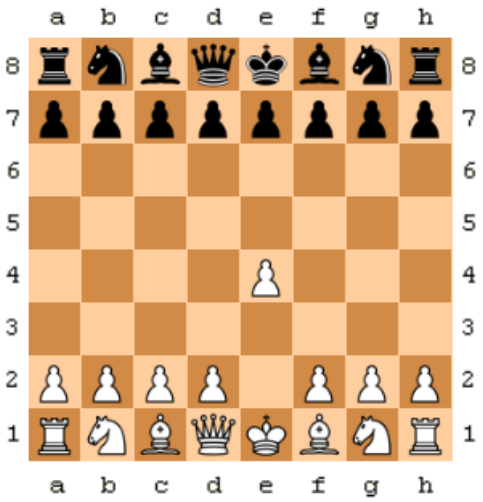


Figura 53. Apertura de peón de rey

Finalmente, las coordenadas transformadas serán las posiciones a las que se moverá el robot, tanto hacia delante como hacia la izquierda o derecha, expresadas en milímetros. Para el caso de la casilla de origen las coordenadas serán: (200,300). Por el lado de la casilla de destino las coordenadas serán: (200,200).

```
IF PuntoOrigen = "e2" THEN
    b := 200;
    a := 300;
ENDIF
```

Figura 54. Casilla de origen

```
IF PuntoDestino = "e4" THEN
    y := 200;
    x := 200;
ENDIF
```

Figura 55. Casilla de destino

4.6.2.2.3 Procedimientos del programa

El programa cuenta con varios procedimientos, los procedimientos son acciones que el robot realizará según las instrucciones que tenga almacenadas dentro de él, de los que nos encontramos los siguientes:

- Main()

- NormalMove()
- AttackMove()
- LongCastlingMove()
- ShortCastlingMove()
- OpenGripper1()
- DropPiece()
- GrippingPiece()
- CloseGripper()

Todos estos procedimientos se explican a continuación:

4.6.2.2.3.1 Main()

Main() es el procedimiento principal del programa, puesto que, contiene todas las instrucciones y procedimientos que serán ejecutados dentro del programa. Este alberga la creación del servidor, la recepción de datos, transformación de escaques a coordenadas, elección de tipo de movimiento y ejecución del mismo. En la se tiene las líneas de programación de este procedimiento.

```
PROC Main()
    createServer;
    Superior:
    ReceiveInformation;
    SquarestoCoord;

    IF Ataque = "False" AND Enroque = "False" THEN
        NormalMove;
        GOTO Superior;
    ENDIF

     IF Ataque = "True" AND Enroque = "False" THEN
        AttackMove;
        GOTO Superior;
    ENDIF

    IF Ataque = "False" AND Enroque = "True" AND PuntoDestino = "c8" THEN
        LongCastlingMove;
        GOTO Superior;
    ENDIF
        IF Ataque = "False" AND Enroque = "True" AND PuntoDestino = "g8" THEN
        ShortCastlingMOve;
        GOTO Superior;
    ENDIF
ENDPROC
```

Figura 56. Procedimiento Main()

4.6.2.2.3.2 NormalMove()

Este procedimiento describe cómo se va a mover el robot en un movimiento regular de pieza. De manera explícita, el robot comienza del punto *Home*, ejecuta el procedimiento *Opengripper1()* (abrir pinza), después se dirige a la casilla de origen, ejecuta *Grippinpiece()* (agarrar pieza), luego se dirige a la casilla de destino, ejecuta el procedimiento *DropPiece()* (soltar pieza), regresa a *Home* y ejecuta *CloseGripper()*.

```
PROC NormalMove()
    MoveJ Home,v500,z100,MyTool\WObj:=wobj0;
    OpenGripper1;
    MoveJ Offs(StartingReference, a, b, safe_height),v500,fine,MyTool\WObj:=WO_tablero;
    MoveL Offs(StartingReference, a, b, 0),v500,fine,MyTool\WObj:=WO_tablero;
    GrippingPiece;
    MoveL Offs(StartingReference, a, b, safe_height),v500,fine,MyTool\WObj:=WO_tablero;
    MoveJ Offs(StartingReference, x, y, safe_height),v500,fine,MyTool\WObj:=WO_tablero;
    MoveL Offs(StartingReference, x, y, 0),v500,fine,MyTool\WObj:=WO_tablero;
    DropPiece;
    MoveL Offs(StartingReference, x, y, safe_height),v500,fine,MyTool\WObj:=WO_tablero;
    MoveJ Home,v500,z100,MyTool\WObj:=wobj0;
    CloseGripper1;
ENDPROC
```

Figura 57. Procedimiento NormalMove()

4.6.2.2.3.3 AttackMove()

En este procedimiento describe el movimiento del robot cuando una pieza captura a otra. El robot comienza en la posición de *Home*, ejecuta *Opengripper1(),* se dirige a la pieza atacada, ejecuta *GrippingPiece()*, la lleva hasta el punto de descarte de piezas (*discard*), ejecuta DropPiece() y después *NormalMove().*

```
PROC AttackMove()
    MoveJ Home,v500,z100,MyTool\WObj:=wobj0;
    OpenGripper1;
    MoveJ Offs(StartingReference, x, y, safe_height),v500,fine,MyTool\WObj:=WO_tablero;
    MoveL Offs(StartingReference, x, y, 0),v500,fine,MyTool\WObj:=WO_tablero;
    GrippingPiece;
    MoveL Offs(StartingReference, x, y, safe_height),v500,fine,MyTool\WObj:=WO_tablero;
    MoveJ Discard,v500,z100,MyTool\WObj:=wobj0;
    DropPiece;
    NormalMove;
ENDPROC
```

Figura 58. Procedimiento AttackMove()

4.6.2.2.3.4 Procedimiento LongCastlingMove() y ShortCastlingMove()

Estos dos procedimientos describen movimientos similares, la única diferencia es que en *LongCastlingMove()* el rey se enroca con la torre del lado de la dama, mientras que en ShortCastlingMove, el rey se enroca con la torre del lado de él. El funcionamiento del robot en este procedimiento es el siguiente: el robot comienza en la posición *Home*, ejecuta *OpenGripper(),* se dirige a la casilla inicial del rey, ejecuta *GrippingPiece()*, lleva al rey a la casilla contigua de la torre, ejecuta *DropPiece(),* se dirige a la casilla de la torre, ejecuta *GrippingPiece()*, lleva a la torre a la izquierda o derecha del rey, según el tipo de enroque (corto o largo), ejecuta *DropPiece() y regresa a Home.*

```
PROC LongCastlingMove()
    MoveJ Home,v500,z100,MyTool\WObj:=wobj0;
    OpenGripper1;
    MoveJ Offs(StartingReference, a, b, safe_height),v500,fine,MyTool\WObj:=WO_tablero;
    MoveL Offs(StartingReference, a, b, 0),v500,fine,MyTool\WObj:=WO_tablero;
    GrippingPiece;
    MoveL Offs(StartingReference, x, y, 0),v500,fine,MyTool\WObj:=WO_tablero;
    DropPiece;
    MoveL Offs(StartingReference, x, y, safe_height),v500,fine,MyTool\WObj:=WO_tablero;
    MoveJ Offs(StartingReference, x, y-100, safe_height),v500,fine,MyTool\WObj:=WO_tablero;
    MoveL Offs(StartingReference, x, y-100, 0),v500,fine,MyTool\WObj:=WO_tablero;
    GrippingPiece;
    MoveL Offs(StartingReference, x, y-100, safe_height),v500,fine,MyTool\WObj:=WO_tablero;
    MoveJ Offs(StartingReference, x, y+50, safe_height),v500,fine,MyTool\WObj:=WO_tablero;
    MoveL Offs(StartingReference, x, y+50, 0),v500,fine,MyTool\WObj:=WO_tablero;
    DropPiece;
    MoveJ Home,v500,z100,MyTool\WObj:=wobj0;
ENDPROC
```

Figura 59. Procedimiento LongCastlingMove()

```
PROC ShortCastlingMove()
    MoveJ Home,v500,z100,MyTool\WObj:=wobj0;
    OpenGripper1;
    MoveJ Offs(StartingReference, a, b, safe_height),v500,fine,MyTool\WObj:=WO_tablero;
    MoveL Offs(StartingReference, a, b, 0),v500,fine,MyTool\WObj:=WO_tablero;
    GrippingPiece;
    MoveL Offs(StartingReference, x, y, 0),v500,fine,MyTool\WObj:=WO_tablero;
    DropPiece;
    MoveL Offs(StartingReference, x, y, safe_height),v500,fine,MyTool\WObj:=WO_tablero;
    MoveJ Offs(StartingReference, x, y+50, safe_height),v500,fine,MyTool\WObj:=WO_tablero;
    MoveL Offs(StartingReference, x, y+50, 0),v500,fine,MyTool\WObj:=WO_tablero;
    GrippingPiece;
    MoveL Offs(StartingReference, x, y+50, safe_height),v500,fine,MyTool\WObj:=WO_tablero;
    MoveJ Offs(StartingReference, x, y-50, safe_height),v500,fine,MyTool\WObj:=WO_tablero;
    MoveL Offs(StartingReference, x, y-50, 0),v500,fine,MyTool\WObj:=WO_tablero;
    DropPiece;
    MoveJ Home,v500,z100,MyTool\WObj:=wobj0;
ENDPROC
```

Figura 60. Procedimiento ShortCastlingMove()

4.6.2.2.3.5 Procedimientos OpenGripper1(), DropPiece(), GrippingPiece() y CloseGripper1()

Todos estos procedimientos tienen injerencia sobre el elemento terminal del robot, en este caso una pinza. En los procedimientos mencionados anteriormente se ejecuta la función *SetDo*, nativa de RobotStudio, que sirve para cambiar el valor de una salida digital, misma que servirá para accionar un actuador (motor), que abrirá y cerrará la pinza según el procedimiento que sea utilizado.

```
PROC OpenGripper1()
    SetDO OpenGripper,1;
    WaitTime 3.5;
    SetDO OpenGripper,0;
ENDPROC

PROC DropPiece()
    SetDO CloseGripper,0;
    SetDO OpenGripper,1;
    WaitTime 1.5;
    SetDO OpenGripper,0;
ENDPROC

PROC GrippingPiece()
   SetDO CloseGripper,1;
   WaitTime 3.0;
   SetDO \SDelay := 2, CloseGripper,0;
ENDPROC

PROC CloseGripper1()
    SetDO CloseGripper,1;
    WaitTime 3.5;
    SetDO\SDelay:=2,CloseGripper,0;
ENDPROC
```

Figura 61. Procedimientos OpenGripper1(), DropPiece(), GrippingPiece() y CloseGripper1().

4.7. Diseño de un elemento terminal para la sujeción de piezas de ajedrez

En esta sección se detallará el diseño de un elemento terminal para la sujeción de piezas de ajedrez. En robótica existe un sinnúmero de aplicaciones en las que se utiliza un brazo robótico y por ende el número de elementos terminales que se utilizan en las mismas es muy elevado. En este trabajo en específico, que es del tipo "*pick and place*" (recoger y colocar), utilizaremos un elemento terminal denominado *gripper* (pinza) para sujetar las piezas y moverlas de una posición a otra. Para la selección del tipo adecuado de gripper se hará un estudio de ellos por: el tipo de agarre, tipo de movimiento, y el tipo de motor eléctrico.

4.7.1. Clasificación de los grippers por su método de agarre

Para hacer frente a las diferentes tareas que se le encomiendan a un elemento terminal, los grippers utilizan diversos métodos de agarre que pueden clasificarse en cuatro grandes grupos:

- **Gripper de impacto.** Se trata de una pinza mecánica en la que la fuerza de sujeción se logra mediante el impacto contra la superficie del objeto desde al menos dos direcciones. Son las más utilizadas en la industria para recoger objetos rígidos utilizando, por ejemplo, pinzas o tenazas.
- **Gripper de ingreso**. Consiste en la penetración de la pieza de trabajo por la herramienta de sujeción. Puede ser intrusiva cuando literalmente penetra en el material, por ejemplo, alfileres, agujas y chinchetas y, por el contrario, puede ser no intrusiva cuando se utilizan otros métodos como el gancho y el bucle, por ejemplo, el velcro. Se utilizan comúnmente con objetos flexibles como los textiles.

- **Gripper astringente**. No es necesario el contacto directo al principio de la sujeción y la fuerza de unión puede tomar la forma de un movimiento de aire para la succión al vacío, el magnetismo o la electroadhesión y se aplica en una sola dirección. Este método de agarre que sólo puede adquirir objetos particulares: para la succión al vacío se necesitan materiales no porosos y rígidos, para la magnetoadhesión se necesitan materiales ferrosos y la electroadhesión sólo es útil para materiales ligeros de chapa y micro componentes.
- **Grippers contiguos**. La superficie del objeto y el medio de sujeción deben hacer contacto directo sin métodos de impactación para producir la fuerza de agarre desde una dirección. Según el tipo de fuerza utilizada, las pinzas contiguas pueden clasificarse en adhesión química como pegamento, adhesión térmica como congelación o fusión y tensión superficial como la acción capilar.

Una vez presentados todos los métodos de agarre, se puede elegir el más adecuado para recoger las piezas de ajedrez. Teniendo en cuenta los objetos no son metálicos, ni de chapa ligera o porosa, el método astrictivo puede ser descartado. De la misma manera, como el ingresivo sólo funciona con tela y las piezas son de plástico, definitivamente no es la mejor opción. Tampoco la pinza contigua es una buena opción debido a las particularidades del método. En conclusión, la mejor opción es usar una pinza de impacto porque es capaz de agarrar todos los objetos mencionados con su versatilidad de formas y materiales.

4.7.2. Grippers de impacto

Las pinzas mecánicas son las más utilizadas en el campo de la industria debido a su gran variedad de aplicaciones. Pueden poseer entre dos y cinco dedos, generalmente con un movimiento sincronizado. Requieren de mecanismos extensos o simples relacionados con los efectos físicos de la mecánica clásica como la amplitud del cono de fricción entre las dos superficies de contacto.

La complejidad de la pinza radica en parte en los grados de libertad, entendiéndola como el número requerido de actuadores independientes que se necesitan para un movimiento completamente definido de todos los enlaces. El más simple sólo requiere un actuador, pero el número de grados de libertad crece con la dificultad de la tarea a realizar.

4.7.2.1. Cinemática

La forma que deben tener los dedos para un propósito determinado se determina estudiando la cinemática del mecanismo. Hay una gran diversidad de diseños para la cadena cinemática con el fin de transformar el movimiento de rotación o de traslación en un movimiento de mandíbula particular. Enfocando en que, las pinzas pueden ser distinguidas:

- **Movimiento paralelo** (Las garras pueden seguir una trayectoria curva o lineal pero siempre permaneciendo paralelos, es decir, sin rotar)
- **Movimiento de rotación** alrededor de un punto fijo
- **Movimiento planar general** de las mandíbulas, por ejemplo, la rotación alrededor de un punto no fijo.

Es esencial conocer la relación de transmisión de la cadena cinemática para controlar el desplazamiento de la mandíbula desde el movimiento motor. La posición de la mandíbula sólo puede ser controlada conociendo la posición del actuador necesario. Esta relación se refleja en la curva característica de la trayectoria de la pinza que da la posición y la orientación de la mandíbula para cada posición del actuador.

Conocer la dependencia de la fuerza de agarre y el par en el motor también es importante a la hora de seleccionar el mecanismo de agarre o incluso el motor apropiado, al menos para asegurarse que es capaz de hacer la fuerza que se requiere.

4.7.2.2. Cadena de transmisión

El primer componente de la cadena de transmisión es siempre el motor, que es el responsable de proporcionar el movimiento de la energía eléctrica. Existen varios tipos de motores en el mercado y para la elección correcta es necesario equilibrar sus características con las necesidades como la precisión en el control de la posición o el par máximo proporcionado. Los siguientes motores pueden ser adecuados:

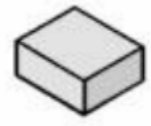
- **Motores paso a paso:** motor eléctrico de corriente continua sin escobillas que divide una rotación completa en un número de pasos iguales. A continuación, se puede ordenar al motor que se mueva y se mantenga en uno de estos pasos sin ningún sensor de retroalimentación (un controlador de bucle abierto). Aplicación en sistemas de bajo costo.
- **Servomotores (motores síncronos):** actuador rotativo que permite un control preciso de la posición angular, la velocidad y la aceleración. Consiste en un motor adecuado acoplado a un sensor para la retroalimentación de la posición. Se aplica cuando se requiere una regulación sensible de la fuerza y la posición.
- **Motores lineales:** un motor eléctrico cuyo estator y rotor se han "desenrollado" para que en lugar de producir un par (rotación) produzca una fuerza lineal a lo largo de su longitud. Aplicable a un funcionamiento proporcional a altas velocidades.
- **Accionamientos piezoeléctricos:** motor eléctrico basado en el cambio de forma de un material piezoeléctrico cuando se aplica un campo eléctrico. Aplicable a objetos extremadamente ligeros y a la manipulación a alta velocidad. Su fiabilidad y vida útil es muy larga, pero la carrera alcanzable es limitada.

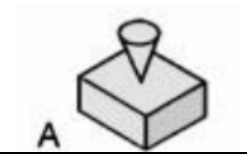
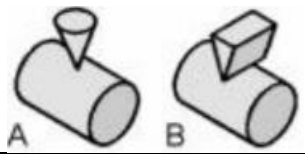
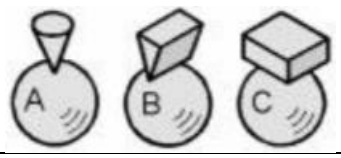
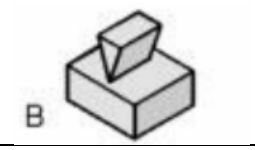
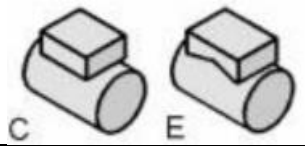
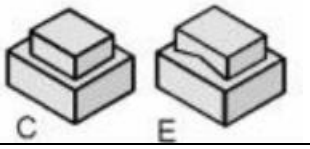
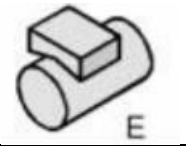
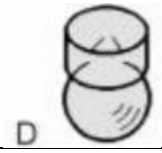
4.7.2.3. Método de contacto

El diseño de las mandíbulas es totalmente determinante para una correcta sujeción ya que es responsable de la distribución de la fuerza de agarre y debe tenerse en cuenta para asegurar la estabilidad.

El movimiento de un objeto en las tres dimensiones del espacio puede ser desagregado en 6 velocidades correspondiente a la rotación y traslación alrededor de los tres ejes. El contacto entre la superficie de la pieza trabajo y el área de agarre de la mandíbula restringen un número específico de esas velocidades (también llamados grados de libertad, k). Un objeto sólo estará completamente sujeto cuando ninguna de sus velocidades es posible.

Tabla 11. La forma de la mandíbula depende de la forma del objeto y del número de grados de libertad que restringe (k)

	Forma del objeto		
	Cúbica	**Cilíndrica**	**Esférica**
k			

1			
2			
3			

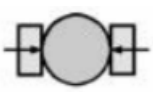
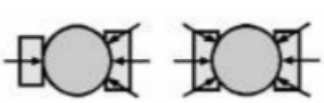
La Tabla 11 ilustra diferentes formas de restringir los grados de libertad k para un cubo, un cilindro y una esfera. Para impedir una velocidad sólo se necesita un punto de contacto, para dos se necesita una línea de contacto o dos puntos de contacto y cualquier otro método de contacto planar restringirá tres velocidades.

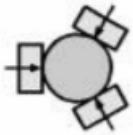
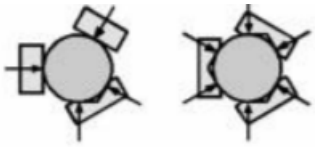
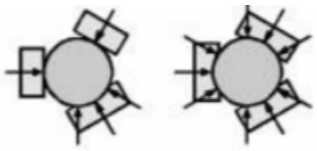
La superficie activa de una pinza es lo que realmente está en contacto entre la mandíbula y el objeto y está relacionada con las formas geométricas utilizadas en los diseños de las mandíbulas. Se designa como: contacto puntual *A*, contacto de línea *B*, contacto de superficie *C*, contacto circular *D*, y contacto de línea doble *E*.

Además de la importancia de la retención total de la pieza, la estabilidad del agarre también debe ser asegurado por la compensación de todas las fuerzas y momentos sobre el objeto. La desalineación de los componentes agarrados no debe ser posible como resultado de su peso o Inercia.

Una reducción de la fuerza de agarre con una mejora de la estabilidad de la retención al mismo tiempo es posible ampliando las superficies activas o aumentándolas en número mediante el uso de más dedos o perfiles más adecuados. En la figura 5 se muestran algunos ejemplos de la combinación de uno a tres dedos y uno, dos o varios puntos de contacto.

Tabla 12. Distribución de la fuerza de sujeción dependiendo el número de puntos de contacto (Gil, Advisor, & Christensen, s. f.)

	Punto de contacto único	**Dos puntos de contacto**	**Múltiples puntos de contacto**
Gripper de 1 dedo			
Gripper de 2 dedos			

Gripper de 3 dedos			

4.7.3. Diseño e implementación

El diseño comienza eligiendo la mejor opción que encaje con todos los requerimientos de la aplicación, en este caso la sujeción de piezas cilíndricas de plástico. Está sección puede ser dividida en la cadena cinemática, el accionamiento y la elección. El prototipo será diseñado en SolidWorks, así como también una simulación de movimiento que brindará algunos datos como el momento necesario para sujetar la pieza, fuerzas de reacción en los dedos y una simulación de esfuerzos a la que estará sometido el gripper.

4.7.3.1. Cadena cinemática

El mecanismo escogido es del tipo biela-manivela, que se utiliza para transformar el movimiento rotacional en movimiento lineal, se utilizarán tres bielas separadas 120º entre sí, que irán conectadas mediante dos articulaciones a los dedos del gripper, que serán los encargados de sujetar las piezas de ajedrez. Las ventajas de elegir este mecanismo son las siguientes:

- Es un diseño de fácil realización.
- Puede agarrar objetos delicados con fuerza y precisión.
- La acción de agarre es rápida y la fuerza es suficientemente fuerte para sostener una pieza estándar de ajedrez.
- Tiene necesidades más bajas de torque cuando agarra objetos pequeños. Tener un torque bajo es relevante, porque se necesitará un motor menos potente.

4.7.3.2. Accionamiento

El actuador escogido es un servomotor, dado que este permite un control preciso de la posición angular, así como también velocidad y aceleración debido a su sensor de retroalimentación. El mercado tiene una gran variedad de servomotores que cambian en especificaciones, tamaño y precio.

El mejor de todos para este caso en particular es *Turnigy TG9e Eco Micro Servo*, con el mismo rendimiento que otros servos con un precio diez veces mayor, sus principales características son:

- **Torque:** 1.5kg/cm (4.8V)
- **Velocidad de operación**: 0.12s/60 grados – 0.10s/60 grados
- **Voltaje de operación:** 4.8V – 6V
- **Engranes:** Plásticos
- **Tipo:** Análogo
- **Rango de temperatura:** 0 – 55 ºC
- **Ancho de banda muerta:** 7µs
- **Peso:** 10.1g
- **Dimensiones:** Figura 62

Servo A(mm)	30.00	Servo B(mm)	23.00
Servo C(mm)	27.00	Servo D(mm)	12.00
Servo E(mm)	33.00	Servo F(mm)	16.00

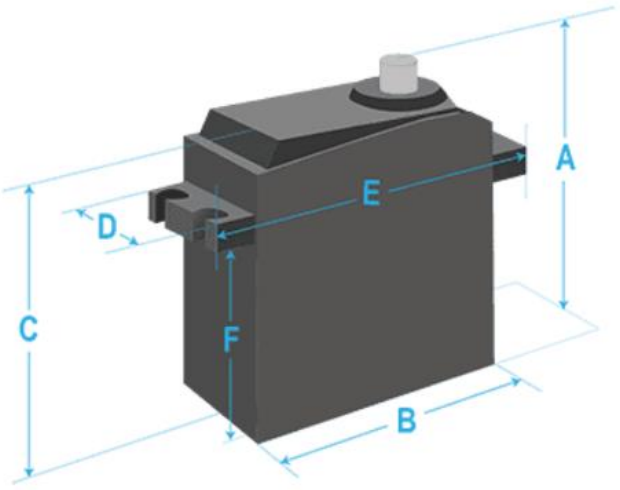


Figura 62. Dimensiones servomotor («Turnigy™ TG9e Eco Micro Servo 1.5kg / 0.10sec / 9g», s. f.)

4.7.3.3. Método de contacto

Para asegurar la estabilidad de las piezas, se necesitará al menos tres puntos de contacto con la misma. Para tener una distribución simétrica de fuerzas, se utilizará un gripper de tres dedos y el diseño debería ser algo similar de lo que se muestra en la Figura 63.

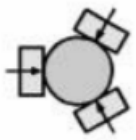


Figura 63. Método de contacto elegido

4.7.4. Prototipo

Como se dijo antes el prototipo consiste en un mecanismo de tres dedos separados entre sí 120º, conectados a una cadena cinemática por tres manivelas curvas, todas ellas conectadas a una biela, misma que va conectada al servomotor. El prototipo del gripper tiene un total de 14 piezas plásticas, que serán construidas en una impresora 3D y tendrán un peso aproximado de 28g. En la Figura 63, se puede ver las principales partes del gripper: 3 dedos, 3 manivelas, 3 articulaciones 1, 3 articulaciones 2, 1 biela, 1 base hexagonal y 1 servomotor.

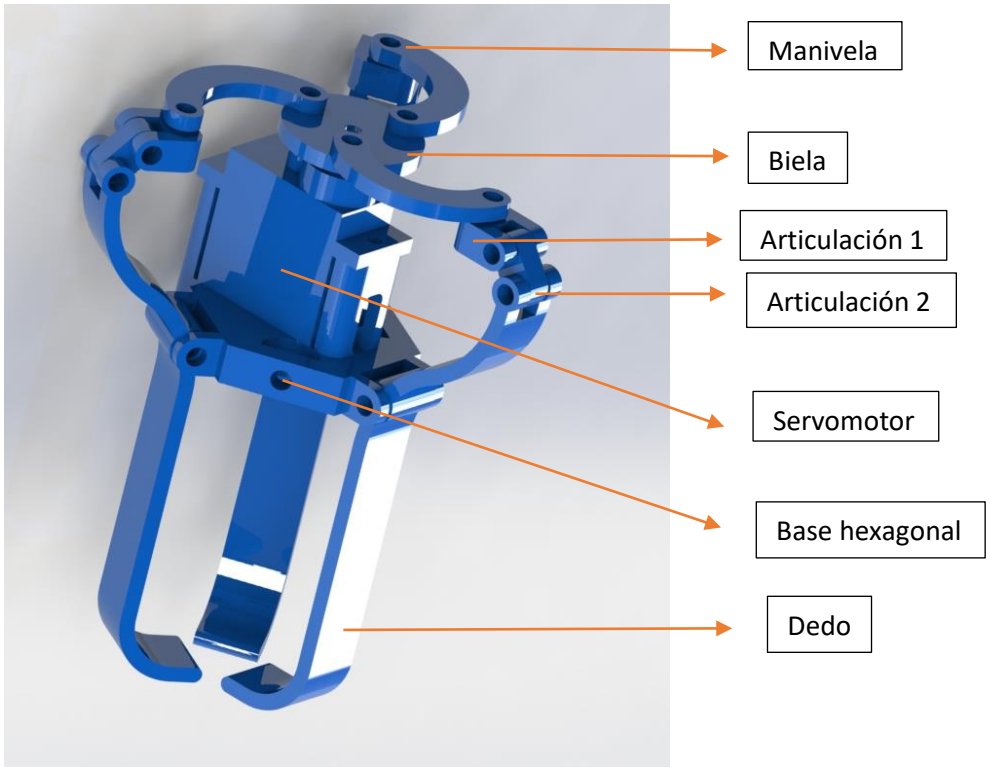


Figura 64. Gripper con sus componentes

El gripper tendrá dos posiciones extremas, las cuales son: totalmente abierta y totalmente cerrada. Las dimensiones de estas dos posiciones son importantes, ya que, en posición abierta debe ser lo suficientemente grande para que entre la pieza y en la posición de cerrada debe ser lo suficientemente pequeña para sostener la pieza. En la Figura 65 tenemos la posición abierta y en la Figura 66 tenemos la posición cerrada. Además, en la Figura 67 se da la dimensión de una de las figuras, en este caso el rey, dado que es la pieza más grande del tablero. Si el rey encaja, las demás piezas lo harán también.

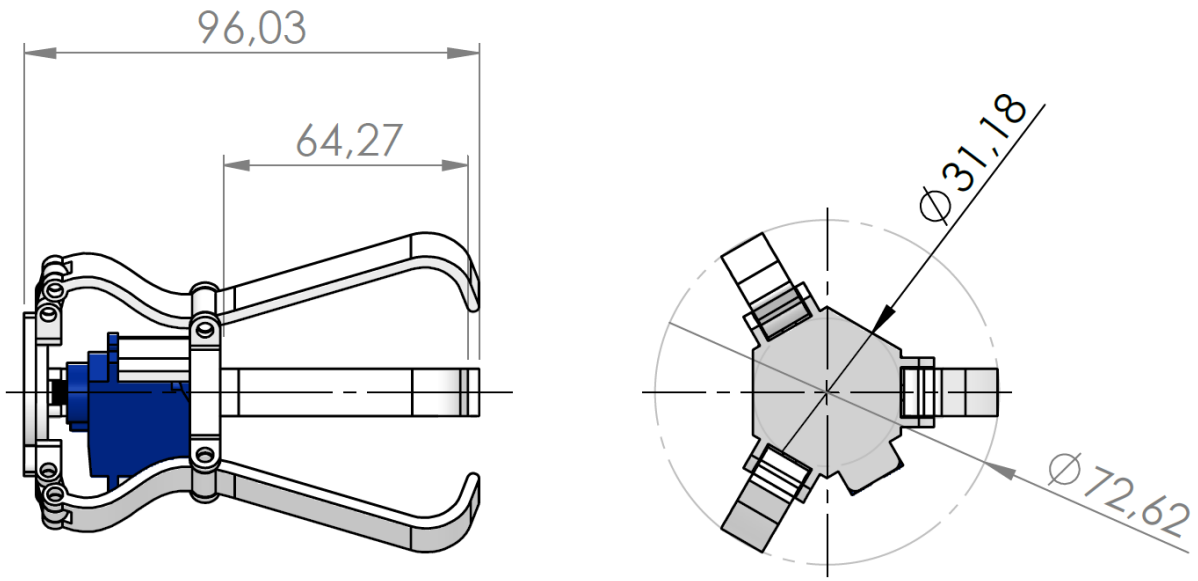


Figura 65. Gripper posición abierta

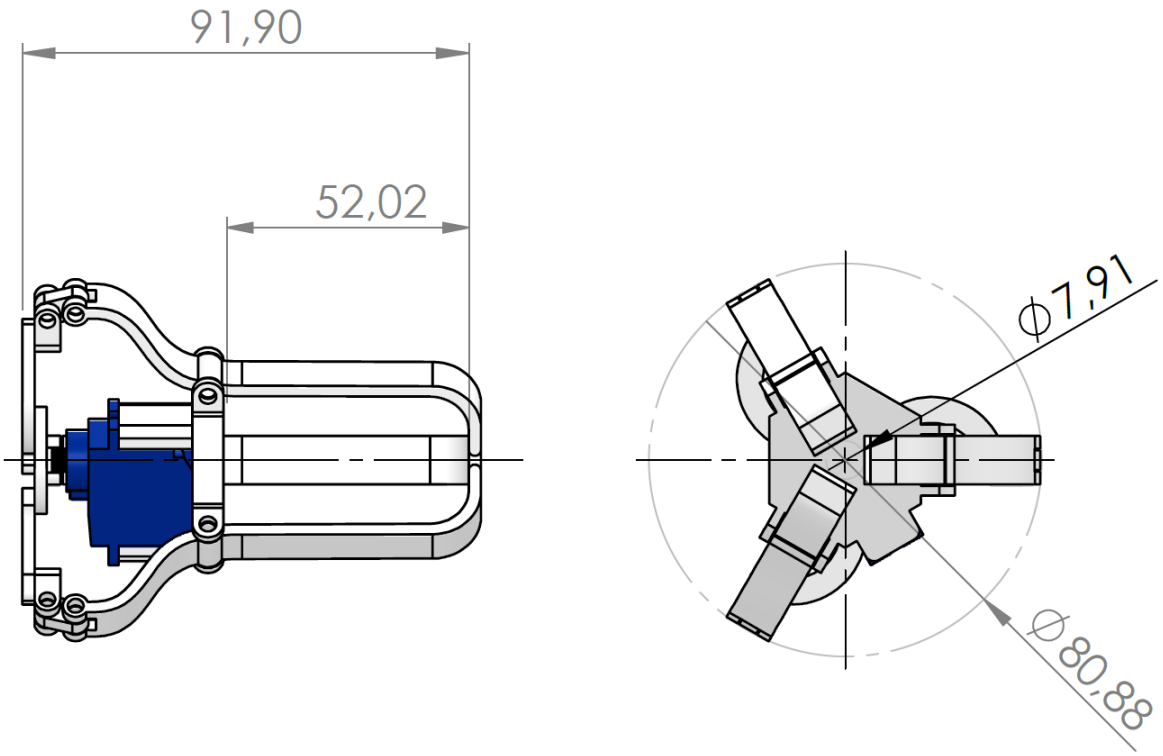


Figura 66. Gripper posición cerrada

Como se puede apreciar en la Figura 67, el rey tiene un diámetro de máximo de 21.32 mm y mínimo de 16.14mm, en su parte superior. Como se indica en las figuras anteriores en posición totalmente abierta la pinza tiene un diámetro máximo de 31.18 mm, por lo que sería suficiente para que entre la pieza. Por otra parte, la pinza en la posición cerrada, tiene un diámetro mínimo de 7.91 mm, entonces sería un diámetro suficientemente chico para sostener la pieza.

Adicionalmente, la longitud desde la punta de la pieza hasta su punto de sujeción es de 42.40 mm, siendo esta medida la mínima que debería tener el interior de la pinza. Esta longitud debe ser tomada cuando la pinza se encuentre cerrada, dado que en esa posición sujetará a la pieza. Finalmente, en las medidas de la pinza cerrada se aprecia que la longitud es de 52.02mm, mayor a los 42.40mm, por lo que la medida sería la adecuada.

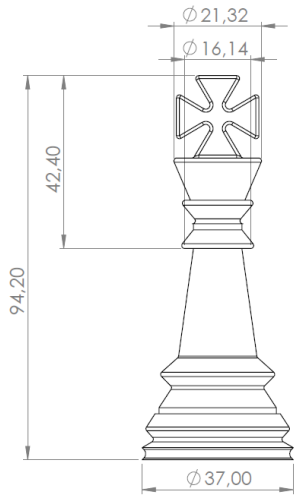


Figura 67. Dimensiones del rey

Para ilustrar de mejor manera el proceso de sujeción observamos las dos posiciones de la pinza (abierta y cerrada), pero esta vez con el rey dentro de ella. Figura 68 y Figura 69.

Figura 68. Inicio de sujeción de pieza

Figura 69. Pieza sujetada

Por último, para pasar al paso de fabricación se debe tener en cuenta la resistencia del material con el que se va a construir el gripper, en este caso será PLA (poliácido láctico), un polímero de uso común en las impresoras 3D. Este material tiene las siguientes características:

Propiedades Generales

- Tiene una densidad de $1{,}24 \times 10^3\ kg/m^3$
- Precio (2,57 - 3,13) EUR/Kg

Propiedades Mecánicas

- Módulo de Young (3,3 - 3,6) GPa
- Coeficiente de Poisson (0,38 - 0,4)
- Límite elástico (55 - 72) MPa
- Resistencia a tracción (47 - 70) MPa
- Resistencia a compresión (66 - 86) MPa
- Elongación (3 - 6) %
- Dureza-Vickers (17 - 27) HV
- Resistencia a fatiga para 10^7 ciclos, (22,2 - 27,7) MPa
- Tenacidad a fractura (3 - 5) MPa*$m^{0.5}$ («Propiedades del PLA», s. f.)

Para determinar cuáles serán las cargas que actuarán sobre los dedos de la pinza, se realizó una simulación de movimiento en Solidworks, tomando en cuenta los datos del servomotor. Después de realizada la simulación se observa que el torque máximo que el motor aplica, cuando está sujetando la pieza, es de 1.1 N-mm (Figura 70). Con la carga que el motor efectúa realizamos un análisis por elementos finitos de uno de los dedos del gripper, obteniéndose un esfuerzo máximo de von Mises de $7.847 \times 10^3\ Pa$ (Figura 71). Si observamos el límite elástico del PLA, 55 MPa, es mucho mayor que el esfuerzo máximo de von Mises, por lo que se puede deducir que el material resistirá perfectamente las cargas a las que será sometido.

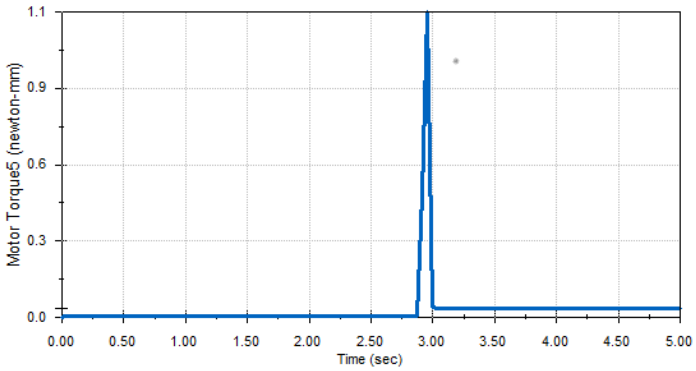


Figura 70. Torque realizado por el servomotor

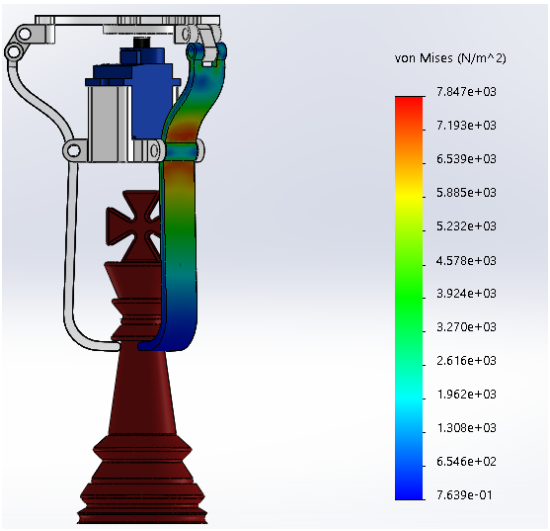


Figura 71. Esfuerzos de von Mises en un dedo del gripper

5. Resultados

El producto final obtenido en este proyecto, consiste en un aplicativo que, por medio de visión artificial por computadora, la ayuda de inteligencia artificial (motor de ajedrez) y el uso de un actuador robótico permite el desarrollo de partidas de ajedrez, hombre-máquina. Las pruebas en el robot físico quedaron supeditadas a la apertura del laboratorio de robótica, sin embargo, los resultados obtenidos por medio de simulación son satisfactorios. A continuación, se brinda un diagrama de flujo completo del sistema de algoritmos de visión y de Inteligencia Artificial aplicados a un robot, para resolver partidas de ajedrez hombre-máquina.

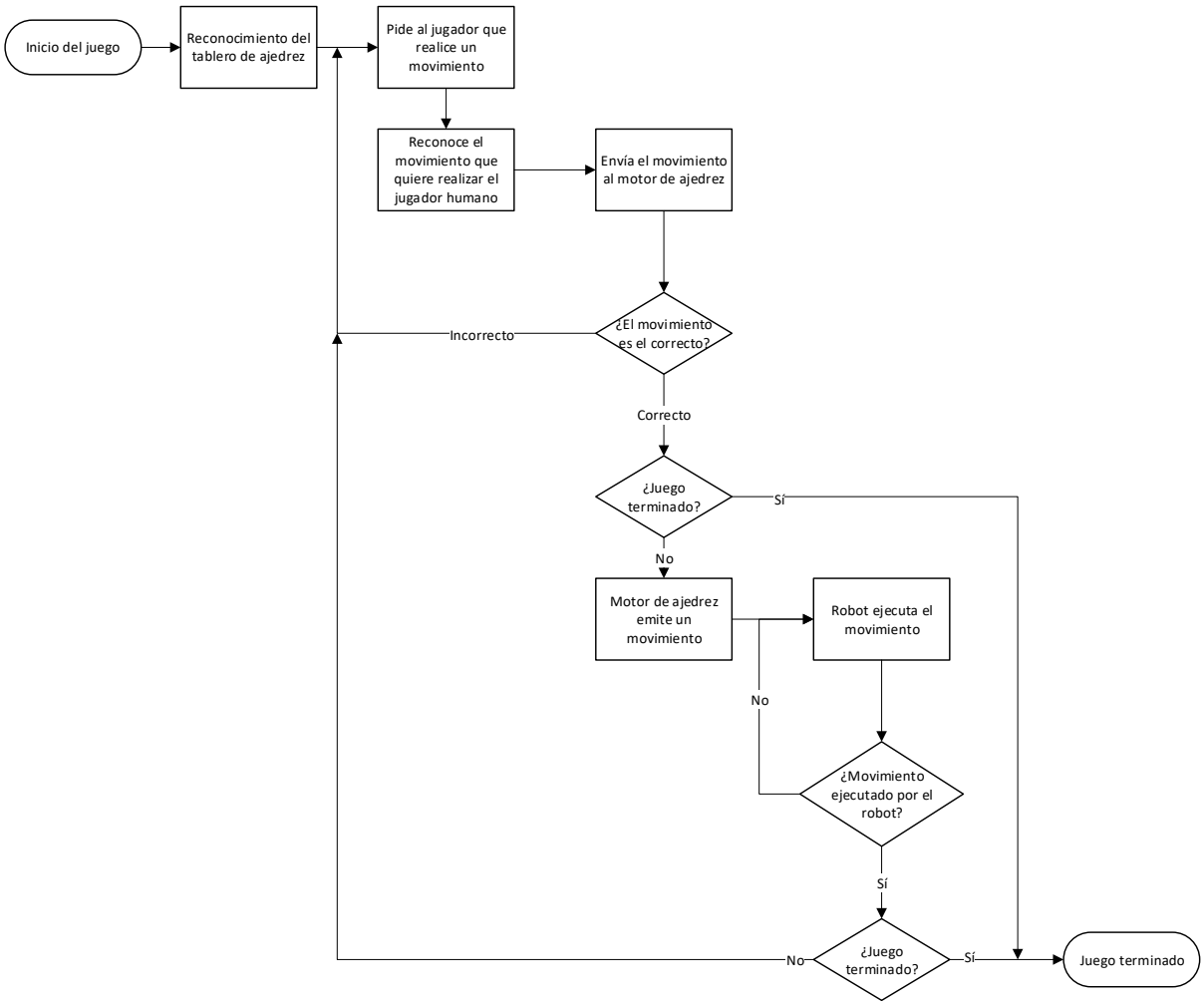


Figura 72. Flujo de trabajo del aplicativo de algoritmos de visión con inteligencia artificial aplicados a un robot, para resolver partidas de ajedrez hombre-maquina

Ahora se expondrá el flujo de trabajo que realiza el programa en un movimiento, esto con el fin de no hacer muy extensa la explicación.

5.1. Reconocimiento del tablero

El programa inicia con el reconocimiento de tablero en el que se solicita una fotografía de mismo, pero sin ninguna pieza. La fotografía tomada del tablero vacío pasa por varios algoritmos que permiten al programa reconocer las líneas, casillas (color) y asigna un nombre a cada una de ellas siguiendo la notación algebraica del ajedrez, la explicación detallada de este paso se encuentra en la sección 4.1.1 de este documento. Seguido de esto, el algoritmo solicita otra fotografía con las piezas colocadas en su lugar para comenzar con el juego. La imagen analizada del tablero queda de la siguiente manera.

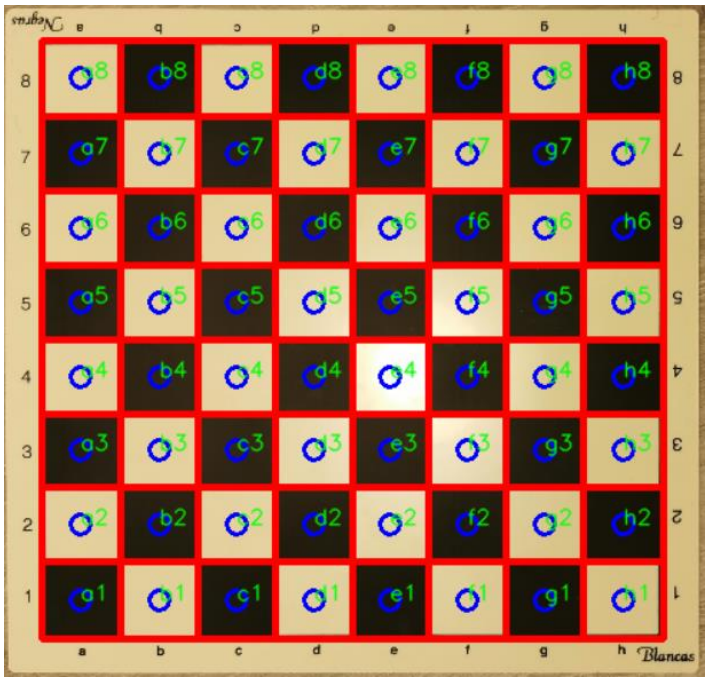


Figura 73. Tablero de ajedrez analizado

5.2. Solicitud de movimiento

Como los movimientos del robot están programados para ser ejecutados en el lado de las piezas negras, el jugador humano siempre dará el primer movimiento, por lo que el programa solicitará que se realice un movimiento para posteriormente ser analizado y determinar que movimiento realizó. En la siguiente figura se ilustra la solicitud que hace el programa.

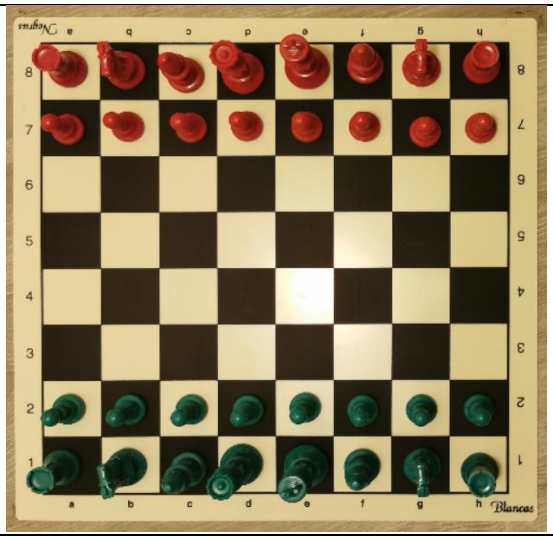
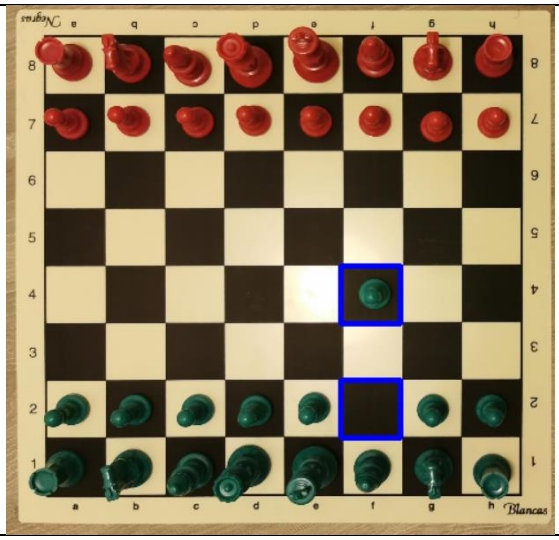
Figura 74. Solicitud de movimiento

5.3. Reconocimiento del movimiento

Después de realizado el movimiento, el aplicativo toma una foto del mismo para comparar las dos imágenes y determinar cuáles fueron los cambios en la misma. En este proceso se observa que casillas

sufrieron el mayor cambio en el color, para de esta manera inferir el movimiento realizado. Este procedimiento se lo puede encontrar con mayor detalle en el apartado 4.1.2. En la tabla inferior se observa el movimiento que realiza la persona y como el programa detecta el mismo.

Tabla 13. Reconocimiento del movimiento realizado

Estado Previo	Estado actual
	

5.4. Enviar el movimiento al motor de ajedrez

Después de que el movimiento es identificado, como el que se observa en la Tabla 13 (f2f4), se lo envía al motor de ajedrez. Para ello, se utiliza una interfaz de comunicación UCI, que transforma este movimiento a un leguaje que el motor de ajedrez pueda entender y de esta manera procese este movimiento y defina si es un movimiento legal o no, para posteriormente emitir una jugada, en caso de que el juego haya terminado comunicará de ello también. La información en detalle se encuentra en la sección 4.2.3.

5.5. Motor de ajedrez emite un movimiento

Una vez el motor de ajedrez haya analizado la jugada realizada por el usuario, emitirá una jugada, este movimiento dependerá de la dificultad de juego seleccionada, es decir, mientras más complejo el juego, el motor de ajedrez mirará más profundamente las posibles jugadas futuras. En el caso que estamos analizando el jugador humano realizó el movimiento *f2f4*. Para esta jugada el motor de ajedrez responde con el movimiento *d7d5*, esto será mostrado en una ventana de la interfaz gráfica de usuario como se muestra en la figura de abajo.

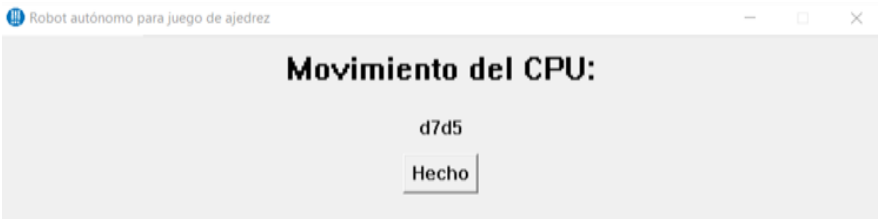
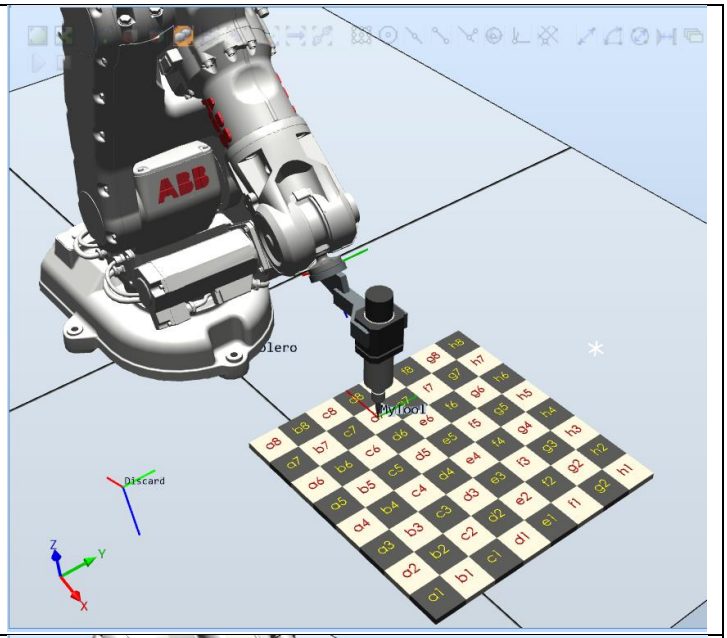
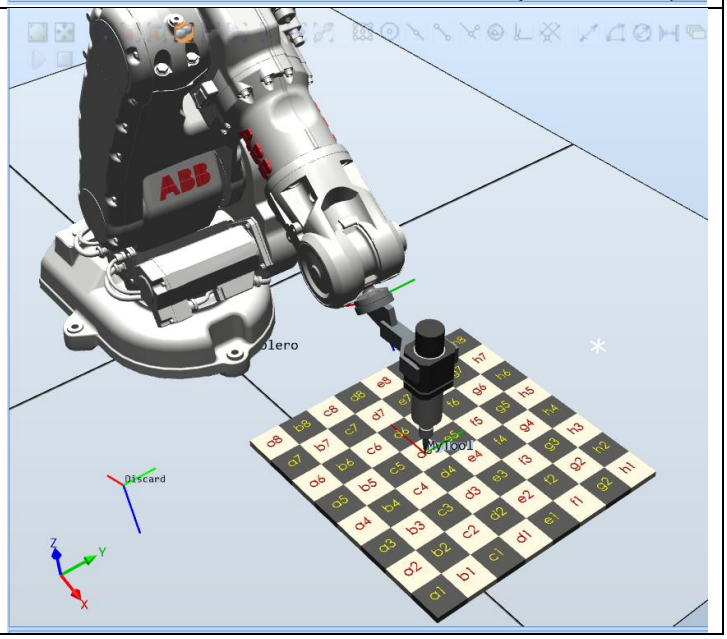


Figura 75. Movimiento emitido por el motor de ajedrez

5.6. Robot ejecuta el movimiento

Cuando el motor de ajedrez emite un movimiento, este es enviado al robot mediante el uso de una interfaz de comunicación, que en este caso es un socket. Después que el movimiento es enviado al robot, el programa que se encuentra dentro de él lo reconoce y lo trasforma en coordenadas que a su vez generan movimiento para trasladar la pieza de una casilla a otra. En la sección 4.6.2 se encuentra la información completa. En la se aprecia el movimiento que realiza el robot que es desde la casilla *d7* a la *d5*.

Tabla 14. Movimiento realizado por el robot

Escaque de origen	
Escaque de destino	

Como este no es un movimiento de jaque mate, el ciclo se repetirá hasta que exista un ganador.

6. Impacto medio ambiental

El gasto energético para la elaboración de este proyecto es muy bajo y por ende las emisiones de CO_2, dado que, para su construcción solo se utilizó un ordenador portátil, que a plena carga consume 120Wh, y una bombilla LED de 14Wh, para iluminación en la noche. Se tomará para el cálculo de emisiones de CO_2 la cantidad de horas que tomó el desarrollo de este proyecto (400 horas), y cuántos kg de CO2 se producen por cada kWh. La Tabla 16 y Tabla 17 ilustran el cálculo desarrollado.

Tabla 15. Consumo de energía en KWh

Fuente de consumo	Potencia [W]	Horas de uso [h]	Consumo total [KWh]
Portátil	120	400	48
Bombilla led	14	200	2,8

Tabla 16. Emisiones de CO_2 en Kg (Ministerio para la transición ecológica y el reto demográfico, s. f.)

KWh consumidos	Kg/CO2 por cada KWh	Emisiones de CO2 [Kg]
50,8	0,31	15,75

Adicionalmente, en lo que respecta al robot, será reciclado cuando haya cumplido su vida útil, alrededor de 35000 horas, por empresas especializadas, que harán uso de sus piezas según convenga. Por último, la cantidad de CO_2 emitido en este proyecto es equiparable al emitido por un coche de gasolina en una semana de uso normal.

7. Presupuesto

El presupuesto que se estima para la creación del presente proyecto se detalla en la Tabla 17.

Tabla 17. Presupuesto general del proyecto

ID	Descripción	Unidad	Valor Unitario	Cantidad	Valor Total
1	Programación del aplicativo de ajedrez por computadora utilizando visión artificial	horas	25,00 €	250	6.250,00 €
2	Programación del robot	horas	25,00 €	75	1.875,00 €
3	Diseño de un elemento terminal (gripper)	horas	25,00 €	75	1.875,00 €
4	Ordenador portátil	unidad	800,00 €	1	800,00 €
5	Tablero de ajedrez plástico 45x45 escaque 50 mm	unidad	5,00 €	1	5,00 €
6	Piezas de ajedrez plásticas de colores	unidad	15,00 €	1	15,00 €
7	Impresión 3D (gripper)	unidad	30,00 €	1	30,00 €
8	TG9e Eco Micro Servo 1.5kg / 0.10sec / 10.1g	unidad	4,00 €	1	4,00 €
				TOTAL	10.854,00 €

Todos los costes asociados a este proyecto son referenciales e incluyen todos los materiales y componentes inmiscuidos en el desarrollo del aplicativo de algoritmo de visión con inteligencia artificial aplicados a un robot, para resolver partidas de ajedrez hombre-máquina.

8. Conclusiones y Trabajo futuro

8.1. Conclusiones

Los objetivos planteados al inicio del proyecto fueron cumplidos en su totalidad, se pasará a determinar cuáles fueron las conclusiones a las que llevó la consecución de los mismos.

El estudio de los principios de procesado de imágenes, permitió el conocimiento de algoritmos de visión por computadora que son utilizados en varios sectores, entre ellos la industria y que son de vital importancia, ya que facilitan las tareas de automatización y control de calidad.

Se diseñó un módulo de visión artificial que permite el reconocimiento del tablero de ajedrez, así como las posiciones en las que se encuentran las piezas, es decir, que se pueda identificar los movimientos realizados por el jugador humano. En las pruebas realizadas a este módulo, se identificó algunas dificultades con la diferenciación de color de las piezas estándar (blancas y negras) y las casillas, motivo por el cual se decidió cambiarlas por unas que tengan un contraste mayor con los escaques (verdes y rojas).

La integración del sistema de reconocimiento de movimientos con el motor de ajedrez se realizó mediante el uso de una interfaz de comunicación conocida como UCI, que permite que los dos módulos se interconecten y transmitan información, el módulo envía movimientos al motor de ajedrez, este analiza si la jugada es legal, si se encuentra en jaque o en jaque mate, también es capaz de devolver un movimiento que posteriormente será ejecutado por un robot.

En torno a la comunicación entre la controladora del robot y el aplicativo, se tenía varias opciones, como, por ejemplo: OPC y Socket, se eligió la última por la simplicidad de su uso. Para establecer la comunicación se utilizó el paradigma de cliente-servidor, en el que el cliente es el aplicativo y el servidor el robot.

La programación de los movimientos del robot se tomó varios aspectos en consideración: área de trabajo del robot, tamaño de las piezas de ajedrez, tamaño del tablero de ajedrez, que velocidad debe tener el robot, ya que interactuará con una persona, tipo de movimiento que realizará. El análisis de estos aspectos permitió establecer cuáles serán los principales puntos de trabajo del robot que en este caso son: el punto de reposo del robot, los centros de las casillas, los extremos del tablero y la altura a la que el robot debe descender para tomar una pieza.

El diseño del elemento terminal fue encaminado al desarrollo y construcción de una pinza “gripper”, que permitirá la sujeción adecuada de las piezas. En este apartado se decidió diseñar una pinza de tres dedos, dada la precisión de agarre que tiene esta y también porque la forma de las piezas es cilíndrica. Para el accionamiento se eligió un servomotor, por el fácil control de la posición de giro y su relativo bajo coste. El método de construcción elegido para esta pinza, es el de impresión 3D, ya que, los esfuerzos que realizará esta son muy bajos.

Finalmente, la conclusión de este proyecto ha ayudado a profundizar temas aprendidos a lo largo del máster como lo es: la programación de robots, con lenguaje RAPID y el uso de nuevas funciones de las que dispone el software de RobotStudio. También se incrementó los conocimientos en herramientas de diseño CAD/CAE, como lo es Solidworks, en el diseño y simulación por elementos finitos y por último se aprendió sobre el lenguaje de programación Python, uno de los lenguajes más utilizados en la actualidad en los campos de ingeniería.

8.2. Trabajo futuro

En el transcurso de la elaboración de este proyecto y las respectivas pruebas que se le hicieron, se han encontrado varias cosas que podrían ser mejoradas para una mejor experiencia y usabilidad del mismo. Todos los puntos susceptibles a mejoras se detallan a continuación:

- Mejoramiento del algoritmo del código de reconocimiento del tablero de ajedrez, ya que el mismo tiene dificultades a la hora de reconocer el tablero de ajedrez si este no se coloca en el centro del foco.
- Colocar un conjunto o arreglo de diodos LEDs encima y en el centro del tablero de ajedrez, de modo que se tenga una distribución homogénea de luz y se evite la producción de sombra que interfieren con el algoritmo de diferencia de color.
- Implementación de un algoritmo de reconocimiento de piezas, para que de esta manera no sea necesario utilizar piezas que tengan un contraste alto con las casillas del tablero. Esto se podría lograr con algoritmo de Deep learning como lo son las redes neuronales convolucionales.
- Mejorar el algoritmo del robot, ya que solo es capaz de jugar de un lado del tablero. Esto se podría resolver agregando todas las posiciones que tendrían las casillas si el robot jugara en el lado opuesto.
- Creación de un tablero virtual que permita la supervisión de las jugadas realizadas por la persona y el robot.
- Crear una opción en la que se pueda controlar al robot de manera manual, esto podría ayudar al desarrollo de partidas de ajedrez en remoto.

Agradecimientos

Quiero expresar mis más sinceros agradecimientos al gobierno ecuatoriano, que, a través de la Secretaría Nacional de Educación, Ciencia y Tecnología, SENESCYT, y a su departamento adscrito, Instituto del Fomento al Talento Humano, IFTH, hicieron posible que continúe y culmine mis estudios de cuarto nivel en el exterior, logrando esta manera alcanzar otro peldaño más en mi vida y seguro que con los estudios realizados podré aportar a mi país de mejor manera.

Adicionalmente, deseo dar las gracias a mis padres, que siempre han estado presentes en los momentos más importantes de mi vida y han sido y serán un pilar fundamental en el cumplimiento de todos los proyectos y metas que me he trazado. Finamente, agradecer de manera especial a todas las personas que han sido mi apoyo en esta estancia en España y que me han brindado su ayuda de manera desinteresada e incondicional.

Bibliografía

ABB. (2004). *Manual de referencia técnica - Instrucciones, funciones y tipos de datos de RAPID*. Recuperado de http://personal.biada.org/~jhorrillo/INSTRUCCIONS RAPID.pdf

ABB. (2007). *Manual del operador Introducción a RAPID*. Recuperado de http://personal.biada.org/~jhorrillo/RAPID Manual operador.pdf

About - Stockfish - Open Source Chess Engine. (s. f.). Recuperado 30 de marzo de 2020, de https://stockfishchess.org/about/

Canny Edge Detection — OpenCV-Python Tutorials 1 documentation. (s. f.). Recuperado 2 de abril de 2020, de https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html

CCRL 40/15 - Index. (s. f.). Recuperado 14 de abril de 2020, de https://ccrl.chessdom.com/ccrl/4040/

Computer Chess Engines: A Quick Guide - Chess.com. (s. f.). Recuperado 14 de abril de 2020, de https://www.chess.com/article/view/computer-chess-engines

Gil, A. M., Advisor, F., & Christensen, D. J. (s. f.). *Gripper design and development for a modular robot*.

Howse, J., & Michino, J. (2020). Learning OpenCV 4 Computer Vision with Python. En *Learning OpenCV 4 Computer Vision with Python 3* (Third). https://doi.org/10.1017/CBO9781107415324.004

Introduction to Python. (s. f.). Recuperado 24 de marzo de 2020, de https://www.w3schools.com/python/python_intro.asp

IRB 140, M2004, Product specification. (s. f.). Recuperado 23 de marzo de 2020, de https://search-ext.abb.com/library/Download.aspx?DocumentID=3HAC041346-001&LanguageCode=en&DocumentPartId=&Action=Launch

Leyes del Ajedrez de la FIDE. (2017). Atenas.

Ministerio para la transición ecológica y el reto demográfico. (s. f.). Factores de emisión. Recuperado 27 de abril de 2020, de https://www.miteco.gob.es/es/cambio-climatico/temas/mitigacion-politicas-y-medidas/factores_emision_tcm30-479095.pdf

Mokrzycki, W., & Tatol, M. (2011). Color difference Delta E - A survey Colour difference Δ E - A survey Faculty of Mathematics and Informatics. *Machine Graphics and Vision*, *20*(4), 383-411. https://doi.org/10.1007/s10817-009-9143-8

OpenCV: cv::VideoCapture Class Reference. (s. f.). Recuperado 1 de abril de 2020, de https://docs.opencv.org/3.4/d8/dfe/classcv_1_1VideoCapture.html

Portátil para juegos Dell G5 de 15 pulgadas para jugadores de nivel medio | Dell España. (s. f.). Recuperado 17 de marzo de 2020, de https://www.dell.com/es-es/shop/portátiles-de-dell/portátil-para-juegos-dell-15-g5/spd/g-series-15-5587-laptop

Propiedades del PLA. (s. f.). Recuperado 22 de abril de 2020, de https://sites.google.com/view/poliacidolactico-coma/poliácido-láctico/propiedades-del-pla

python-chess: a pure Python chess library — python-chess 0.30.1 documentation. (s. f.). Recuperado 9 de abril de 2020, de https://python-chess.readthedocs.io/en/latest/

Python | Thresholding techniques using OpenCV | Set-2 (Adaptive Thresholding) - GeeksforGeeks. (s. f.). Recuperado 1 de abril de 2020, de https://www.geeksforgeeks.org/python-thresholding-techniques-using-opencv-set-2-adaptive-thresholding/

Python OpenCV | cv2.cvtColor() method - GeeksforGeeks. (s. f.). Recuperado 1 de abril de 2020, de https://www.geeksforgeeks.org/python-opencv-cv2-cvtcolor-method/

Robotics, A. (2004). *Especificaciones del producto - Controlador IRC5 con FlexPendant*.

Robotics, A. (2019). *Manual del operador - RobotStudio*.

RobotStudio - ABB Robotics. (s. f.). Recuperado 30 de marzo de 2020, de https://new.abb.com/products/robotics/robotstudio

Turnigy™ TG9e Eco Micro Servo 1.5kg / 0.10sec / 9g. (s. f.). Recuperado 22 de abril de 2020, de https://hobbyking.com/en_us/turnigytm-tg9e-eco-micro-servo-1-5kg-0-10sec-9g.html?___store=en_us

What is PyCharm? | What is Pycharm used for? | Intellipaat Blog. (s. f.). Recuperado 26 de marzo de 2020, de https://intellipaat.com/blog/what-is-pycharm/

What is Python? Executive Summary | Python.org. (s. f.). Recuperado 24 de marzo de 2020, de https://www.python.org/doc/essays/blurb/

What is SOLIDWORKS? | Capitol Technology University. (s. f.). Recuperado 30 de marzo de 2020, de https://www.captechu.edu/blog/solidworks-mechatronics-design-and-engineering-program

ANEXOS

Código programa RobotStudio

Creación de módulo principal

```
MODULE Module1
   CONST robtarget
Home:=[[689.268487273,0,537.14730631],[0.190808996,0,0.981627183,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9
E+09]];
   CONST robtarget StartingReference:=[[425,-175,210],[0,0,1,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
   CONST robtarget Reference2:=[[800,200,210],[0,0,1,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
   CONST robtarget Reference1:=[[400,-200,210],[0,0,1,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
   CONST num safe_height := 110;
   CONST robtarget Discard:=[[500,-400,250],[0.190808995,0,0.981627183,0],[-1,0,-
2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];


        PROC Main()
    createServer;
    Superior:
    ReceiveInformation;
    SquarestoCoord;

    IF Ataque = "False" AND Enroque = "False" THEN
       NormalMove;
       GOTO Superior;
    ENDIF

     IF Ataque = "True" AND Enroque = "False" THEN
       AttackMove;
       GOTO Superior;
    ENDIF

    IF Ataque = "False" AND Enroque = "True" AND PuntoDestino = "c8" THEN
       LongCastlingMove;
       GOTO Superior;
    ENDIF
       IF Ataque = "False" AND Enroque = "True" AND PuntoDestino = "g8" THEN
       ShortCastlingMOve;
       GOTO Superior;
    ENDIF
  ENDPROC

  PROC NormalMove()
    MoveJ Home,v100,z100,MyTool\WObj:=wobj0;
    OpenGripper1;
    MoveJ Offs(StartingReference, a, b, safe_height),v100,fine,MyTool\WObj:=WO_tablero;
    MoveL Offs(StartingReference, a, b, 0),v100,fine,MyTool\WObj:=WO_tablero;
    GrippingPiece;
    MoveL Offs(StartingReference, a, b, safe_height),v100,fine,MyTool\WObj:=WO_tablero;
    MoveJ Offs(StartingReference, x, y, safe_height),v100,fine,MyTool\WObj:=WO_tablero;
    MoveL Offs(StartingReference, x, y, 0),v100,fine,MyTool\WObj:=WO_tablero;
    DropPiece;
    MoveL Offs(StartingReference, x, y, safe_height),v100,fine,MyTool\WObj:=WO_tablero;
    MoveJ Home,v100,z100,MyTool\WObj:=wobj0;
    CloseGripper1;
  ENDPROC

  PROC AttackMove()
    MoveJ Home,v100,z100,MyTool\WObj:=wobj0;
    OpenGripper1;
    MoveJ Offs(StartingReference, x, y, safe_height),v100,fine,MyTool\WObj:=WO_tablero;
```

```
    MoveL Offs(StartingReference, x, y, 0),v100,fine,MyTool\WObj:=WO_tablero;
    GrippingPiece;
    MoveL Offs(StartingReference, x, y, safe_height),v100,fine,MyTool\WObj:=WO_tablero;
    MoveJ Discard,v100,z100,MyTool\WObj:=wobj0;
    DropPiece;
    NormalMove;
ENDPROC

PROC LongCastlingMove()
    MoveJ Home,v100,z100,MyTool\WObj:=wobj0;
    OpenGripper1;
    MoveJ Offs(StartingReference, a, b, safe_height),v100,fine,MyTool\WObj:=WO_tablero;
    MoveL Offs(StartingReference, a, b, 0),v100,fine,MyTool\WObj:=WO_tablero;
    GrippingPiece;
    MoveL Offs(StartingReference, x, y, 0),v100,fine,MyTool\WObj:=WO_tablero;
    DropPiece;
    MoveL Offs(StartingReference, x, y, safe_height),v100,fine,MyTool\WObj:=WO_tablero;
    MoveJ Offs(StartingReference, x, y-100, safe_height),v100,fine,MyTool\WObj:=WO_tablero;
    MoveL Offs(StartingReference, x, y-100, 0),v100,fine,MyTool\WObj:=WO_tablero;
    GrippingPiece;
    MoveL Offs(StartingReference, x, y-100, safe_height),v100,fine,MyTool\WObj:=WO_tablero;
    MoveJ Offs(StartingReference, x, y+50, safe_height),v100,fine,MyTool\WObj:=WO_tablero;
    MoveL Offs(StartingReference, x, y+50, 0),v100,fine,MyTool\WObj:=WO_tablero;
    DropPiece;
    MoveJ Home,v100,z100,MyTool\WObj:=wobj0;
ENDPROC

PROC ShortCastlingMove()
    MoveJ Home,v100,z100,MyTool\WObj:=wobj0;
    OpenGripper1;
    MoveJ Offs(StartingReference, a, b, safe_height),v100,fine,MyTool\WObj:=WO_tablero;
    MoveL Offs(StartingReference, a, b, 0),v100,fine,MyTool\WObj:=WO_tablero;
    GrippingPiece;
    MoveL Offs(StartingReference, x, y, 0),v100,fine,MyTool\WObj:=WO_tablero;
    DropPiece;
    MoveL Offs(StartingReference, x, y, safe_height),v100,fine,MyTool\WObj:=WO_tablero;
    MoveJ Offs(StartingReference, x, y+50, safe_height),v100,fine,MyTool\WObj:=WO_tablero;
    MoveL Offs(StartingReference, x, y+50, 0),v100,fine,MyTool\WObj:=WO_tablero;
    GrippingPiece;
    MoveL Offs(StartingReference, x, y+50, safe_height),v100,fine,MyTool\WObj:=WO_tablero;
    MoveJ Offs(StartingReference, x, y-50, safe_height),v100,fine,MyTool\WObj:=WO_tablero;
    MoveL Offs(StartingReference, x, y-50, 0),v100,fine,MyTool\WObj:=WO_tablero;
    DropPiece;
    MoveJ Home,v100,z100,MyTool\WObj:=wobj0;


ENDPROC

PROC OpenGripper1()

    SetDO OpenGripper,1;
    WaitTime 3.5;
    SetDO OpenGripper,0;

ENDPROC

PROC DropPiece()

    SetDO CloseGripper,0;
    SetDO OpenGripper,1;
    WaitTime 1.5;
    SetDO OpenGripper,0;

ENDPROC
```

```
PROC GrippingPiece()

  SetDO CloseGripper,1;
  WaitTime 3.0;
  SetDO \SDelay := 2, CloseGripper,0;

ENDPROC
PROC CloseGripper1()
   SetDO CloseGripper,1;
   WaitTime 3.5;
   SetDO\SDelay:=2,CloseGripper,0;
ENDPROC


ENDMODULE
```

Creación del servidor y de las posiciones

```
MODULE MOD1
  !Declaración de variables
  VAR socketdev server;
  VAR socketdev client;
  VAR string message;
  VAR num coordenada;
  VAR string receiveString;
  VAR string PuntoOrigen;
  VAR string PuntoDestino;
  VAR string Ataque;
  VAR string Enroque;
  VAR num count:=0;
  VAR num a;
  VAR num b;
  VAR num x;
  VAR num y;

  PROC createServer()

      ! Crea la comunicación
      SocketCreate server;
      SocketBind server,"127.0.0.1",55000;
      SocketListen server;
      SocketAccept server,client\Time:=WAIT_MAX;

  ENDPROC

  PROC ReceiveInformation()

      FOR count FROM 0 TO 3 DO
        IF count = 0 THEN
         SocketReceive client,\Str:=receiveString\Time:=WAIT_MAX;
         PuntoOrigen:=receiveString;
         SocketSend client,\Str:="Punto de origen";
         ELSEIF count = 1 THEN
         SocketReceive client,\Str:=receiveString;
         PuntoDestino:=receiveString;
         SocketSend client,\Str:="Punto de destino";
         ELSEIF count = 2 THEN
         SocketReceive client,\Str:=receiveString;
         Ataque:=receiveString;
         SocketSend client,\Str:="Ataque";
         ELSEIF count = 3 THEN
         SocketReceive client,\Str:=receiveString;
         Enroque:=receiveString;
         SocketSend client,\Str:="Enroque";
```

```
        ENDIF
      ENDFOR
    count:=0;


ERROR
    IF ERRNO=ERR_SOCK_TIMEOUT THEN
      RETRY;
    ELSEIF ERRNO=ERR_SOCK_CLOSED THEN
      RETURN ;
    ELSE
      ! Sin manejo de recuperación de errores
    ENDIF

    ! Cerrar comunicación
    SocketClose server;


ENDPROC

PROC SquarestoCoord()

    ! FILA 8

    IF PuntoOrigen = "a8" THEN
      b := 0;
      a := 0;
    ENDIF

    IF PuntoOrigen = "b8" THEN
      b := 50;
      a := 0;
    ENDIF

    IF PuntoOrigen = "c8" THEN
      b := 100;
      a := 0;
    ENDIF

    IF PuntoOrigen = "d8" THEN
      b := 150;
      a := 0;
    ENDIF

    IF PuntoOrigen = "e8" THEN
      b := 200;
      a := 0;
    ENDIF

    IF PuntoOrigen = "f8" THEN
      b := 250;
      a := 0;
    ENDIF

    IF PuntoOrigen = "g8" THEN
      b := 300;
      a := 0;
    ENDIF

    IF PuntoOrigen = "h8" THEN
      b := 350;
      a := 0;
    ENDIF
```

```
IF PuntoDestino = "a8" THEN
   y := 0;
   x := 0;
ENDIF

IF PuntoDestino = "b8" THEN
   y := 50;
   x := 0;
ENDIF

IF PuntoDestino = "c8" THEN
   y := 100;
   x := 0;
ENDIF

IF PuntoDestino = "d8" THEN
   y := 150;
   x := 0;
ENDIF

IF PuntoDestino = "e8" THEN
   y := 200;
   x := 0;
ENDIF

IF PuntoDestino = "f8" THEN
   y := 250;
   x := 0;
ENDIF

IF PuntoDestino = "g8" THEN
   y := 300;
   x := 0;
ENDIF

IF PuntoDestino = "h8" THEN
   y := 350;
   x := 0;
ENDIF

! FILA 7

IF PuntoOrigen = "a7" THEN
   b := 0;
   a := 50;
ENDIF

IF PuntoOrigen = "b7" THEN
   b := 50;
   a := 50;
ENDIF

IF PuntoOrigen = "c7" THEN
   b := 100;
   a := 50;
ENDIF

IF PuntoOrigen = "d7" THEN
   b := 150;
   a := 50;
ENDIF

IF PuntoOrigen = "e7" THEN
   b := 200;
   a := 50;
ENDIF
```

```
IF PuntoOrigen = "f7" THEN
   b := 250;
   a := 50;
ENDIF

IF PuntoOrigen = "g7" THEN
   b := 300;
   a := 50;
ENDIF

IF PuntoOrigen = "h7" THEN
   b := 350;
   a := 50;
ENDIF

IF PuntoDestino = "a7" THEN
   y := 0;
   x := 50;
ENDIF

IF PuntoDestino = "b7" THEN
   y := 50;
   x := 50;
ENDIF

IF PuntoDestino = "c7" THEN
   y := 100;
   x := 50;
ENDIF

IF PuntoDestino = "d7" THEN
   y := 150;
   x := 50;
ENDIF

IF PuntoDestino = "e7" THEN
   y := 200;
   x := 50;
ENDIF

IF PuntoDestino = "f7" THEN
   y := 250;
   x := 50;
ENDIF

IF PuntoDestino = "g7" THEN
   y := 300;
   x := 50;
ENDIF

IF PuntoDestino = "h7" THEN
   y := 350;
   x := 50;
ENDIF

! FILA 6

IF PuntoOrigen = "a6" THEN
   b := 0;
   a := 100;
ENDIF

IF PuntoOrigen = "b6" THEN
   b := 50;
   a := 100;
```

```
ENDIF

IF PuntoOrigen = "c6" THEN
   b := 100;
   a := 100;
ENDIF

IF PuntoOrigen = "d6" THEN
   b := 150;
   a := 100;
ENDIF

IF PuntoOrigen = "e6" THEN
   b := 200;
   a := 100;
ENDIF

IF PuntoOrigen = "f6" THEN
   b := 250;
   a := 100;
ENDIF

IF PuntoOrigen = "g6" THEN
   b := 300;
   a := 100;
ENDIF

IF PuntoOrigen = "h6" THEN
   b := 350;
   a := 100;
ENDIF

IF PuntoDestino = "a6" THEN
   y := 0;
   x := 100;
ENDIF

IF PuntoDestino = "b6" THEN
   y := 50;
   x := 100;
ENDIF

IF PuntoDestino = "c6" THEN
   y := 100;
   x := 100;
ENDIF

IF PuntoDestino = "d6" THEN
   y := 150;
   x := 100;
ENDIF

IF PuntoDestino = "e6" THEN
   y := 200;
   x := 100;
ENDIF

IF PuntoDestino = "f6" THEN
   y := 250;
   x := 100;
ENDIF

IF PuntoDestino = "g6" THEN
   y := 300;
   x := 100;
ENDIF
```

```
IF PuntoDestino = "h6" THEN
   y := 350;
   x := 100;
ENDIF

! FILA 5

IF PuntoOrigen = "a5" THEN
   b := 0;
   a := 150;
ENDIF

IF PuntoOrigen = "b5" THEN
   b := 50;
   a := 150;
ENDIF

IF PuntoOrigen = "c5" THEN
   b := 100;
   a := 150;
ENDIF

IF PuntoOrigen = "d5" THEN
   b := 150;
   a := 150;
ENDIF

IF PuntoOrigen = "e5" THEN
   b := 200;
   a := 150;
ENDIF

IF PuntoOrigen = "f5" THEN
   b := 250;
   a := 150;
ENDIF

IF PuntoOrigen = "g5" THEN
   b := 300;
   a := 150;
ENDIF

IF PuntoOrigen = "h5" THEN
   b := 350;
   a := 150;
ENDIF

IF PuntoDestino = "a5" THEN
   y := 0;
   x := 150;
ENDIF

IF PuntoDestino = "b5" THEN
   y := 50;
   x := 150;
ENDIF

IF PuntoDestino = "c5" THEN
   y := 100;
   x := 150;
ENDIF

IF PuntoDestino = "d5" THEN
   y := 150;
   x := 150;
```

```
ENDIF

IF PuntoDestino = "e5" THEN
   y := 200;
   x := 150;
ENDIF

IF PuntoDestino = "f5" THEN
   y := 250;
   x := 150;
ENDIF

IF PuntoDestino = "g5" THEN
   y := 300;
   x := 150;
ENDIF

IF PuntoDestino = "h5" THEN
   y := 350;
   x := 150;
ENDIF

! FILA 4

IF PuntoOrigen = "a4" THEN
   b := 0;
   a := 200;
ENDIF

IF PuntoOrigen = "b4" THEN
   b := 50;
   a := 200;
ENDIF

IF PuntoOrigen = "c4" THEN
   b := 100;
   a := 200;
ENDIF

IF PuntoOrigen = "d4" THEN
   b := 150;
   a := 200;
ENDIF

IF PuntoOrigen = "e4" THEN
   b := 200;
   a := 200;
ENDIF

IF PuntoOrigen = "f4" THEN
   b := 250;
   a := 200;
ENDIF

IF PuntoOrigen = "g4" THEN
   b := 300;
   a := 200;
ENDIF

IF PuntoOrigen = "h4" THEN
   b := 350;
   a := 200;
ENDIF

IF PuntoDestino = "a4" THEN
   y := 0;
```

```
    x := 200;
ENDIF

IF PuntoDestino = "b4" THEN
    y := 50;
    x := 200;
ENDIF

IF PuntoDestino = "c4" THEN
    y := 100;
    x := 200;
ENDIF

IF PuntoDestino = "d4" THEN
    y := 150;
    x := 200;
ENDIF

IF PuntoDestino = "e4" THEN
    y := 200;
    x := 200;
ENDIF

IF PuntoDestino = "f4" THEN
    y := 250;
    x := 200;
ENDIF

IF PuntoDestino = "g4" THEN
    y := 300;
    x := 200;
ENDIF

IF PuntoDestino = "h4" THEN
    y := 350;
    x := 200;
ENDIF

! FILA 3

IF PuntoOrigen = "a3" THEN
    b := 0;
    a := 250;
ENDIF

IF PuntoOrigen = "b3" THEN
    b := 50;
    a := 250;
ENDIF

IF PuntoOrigen = "c3" THEN
    b := 100;
    a := 250;
ENDIF

IF PuntoOrigen = "d3" THEN
    b := 150;
    a := 250;
ENDIF

IF PuntoOrigen = "e3" THEN
    b := 200;
    a := 250;
ENDIF

IF PuntoOrigen = "f3" THEN
```

```
    b := 250;
    a := 250;
ENDIF

IF PuntoOrigen = "g3" THEN
    b := 300;
    a := 250;
ENDIF

IF PuntoOrigen = "h3" THEN
    b := 350;
    a := 250;
ENDIF

IF PuntoDestino = "a3" THEN
    y := 0;
    x := 250;
ENDIF

IF PuntoDestino = "b3" THEN
    y := 50;
    x := 250;
ENDIF

IF PuntoDestino = "c3" THEN
    y := 100;
    x := 250;
ENDIF

IF PuntoDestino = "d3" THEN
    y := 150;
    x := 250;
ENDIF

IF PuntoDestino = "e3" THEN
    y := 200;
    x := 250;
ENDIF

IF PuntoDestino = "f3" THEN
    y := 250;
    x := 250;
ENDIF

IF PuntoDestino = "g3" THEN
    y := 300;
    x := 250;
ENDIF

IF PuntoDestino = "h3" THEN
    y := 350;
    x := 250;
ENDIF

! FILA 2

IF PuntoOrigen = "a2" THEN
    b := 0;
    a := 300;
ENDIF

IF PuntoOrigen = "b2" THEN
    b := 50;
    a := 300;
ENDIF
```

```
IF PuntoOrigen = "c2" THEN
   b := 100;
   a := 300;
ENDIF

IF PuntoOrigen = "d2" THEN
   b := 150;
   a := 300;
ENDIF

IF PuntoOrigen = "e2" THEN
   b := 200;
   a := 300;
ENDIF

IF PuntoOrigen = "f2" THEN
   b := 250;
   a := 300;
ENDIF

IF PuntoOrigen = "g2" THEN
   b := 300;
   a := 300;
ENDIF

IF PuntoOrigen = "h2" THEN
   b := 350;
   a := 300;
ENDIF

IF PuntoDestino = "a2" THEN
   y := 0;
   x := 300;
ENDIF

IF PuntoDestino = "b2" THEN
   y := 50;
   x := 300;
ENDIF

IF PuntoDestino = "c2" THEN
   y := 100;
   x := 300;
ENDIF

IF PuntoDestino = "d2" THEN
   y := 150;
   x := 300;
ENDIF

IF PuntoDestino = "e2" THEN
   y := 200;
   x := 300;
ENDIF

IF PuntoDestino = "f2" THEN
   y := 250;
   x := 300;
ENDIF

IF PuntoDestino = "g2" THEN
   y := 300;
   x := 300;
ENDIF

IF PuntoDestino = "h2" THEN
```

```
    y := 350;
    x := 300;
ENDIF

! FILA 8

IF PuntoOrigen = "a1" THEN
    b := 0;
    a := 350;
ENDIF

IF PuntoOrigen = "b1" THEN
    b := 50;
    a := 350;
ENDIF

IF PuntoOrigen = "c1" THEN
    b := 100;
    a := 350;
ENDIF

IF PuntoOrigen = "d1" THEN
    b := 150;
    a := 350;
ENDIF

IF PuntoOrigen = "e1" THEN
    b := 200;
    a := 350;
ENDIF

IF PuntoOrigen = "f1" THEN
    b := 250;
    a := 350;
ENDIF

IF PuntoOrigen = "g1" THEN
    b := 300;
    a := 350;
ENDIF

IF PuntoOrigen = "h1" THEN
    b := 350;
    a := 350;
ENDIF

IF PuntoDestino = "a1" THEN
    y := 0;
    x := 350;
ENDIF

IF PuntoDestino = "b1" THEN
    y := 50;
    x := 350;
ENDIF

IF PuntoDestino = "c1" THEN
    y := 100;
    x := 350;
ENDIF

IF PuntoDestino = "d1" THEN
    y := 150;
    x := 350;
ENDIF
```

```
    IF PuntoDestino = "e1" THEN
        y := 200;
        x := 350;
    ENDIF

    IF PuntoDestino = "f1" THEN
        y := 250;
        x := 350;
    ENDIF

    IF PuntoDestino = "g1" THEN
        y := 300;
        x := 350;
    ENDIF

    IF PuntoDestino = "h1" THEN
        y := 350;
        x := 350;
    ENDIF

  ENDPROC

ENDMODULE
```

Código de programación Python

Cámara

```
import cv2

class Camera:

   def get_frames(self):

      capture = cv2.VideoCapture(1)
      capture.set(3, 1280)
      capture.set(4, 720)

      if not capture.isOpened():
         raise RuntimeError('Fallo inicio de cámara')

      while True:
         ret, img = capture.read()
         cv2.imshow("Original", img)
         if cv2.waitKey(5) & 0xFF == ord('q'):
            break
      cv2.destroyAllWindows()
      capture.release()
      return img
```

Línea

```
class Line:

   def __init__(self,x1,x2,y1,y2):
      '''
      Crea el objeto línea
      '''

      # Punto finales
      self.x1 = x1
      self.x2 = x2
      self.y1 = y1
      self.y2 = y2


      # Cambios en x e y
      self.dx = self.x2 - self.x1
      self.dy = self.y2 - self.y1

      # Orientación
      if abs(self.dx) > abs(self.dy):
         self.orientation = 'horizontal'
      else:
         self.orientation = 'vertical'


   def find_intersection(self,other):
      '''
      Encuentra la intersección de esta línea y otra. Una línea debe ser
horizontal
      y la otra debe ser vertical.


      # Determinante para hallar los puntos de intersección de las líneas
      x = ((self.x1*self.y2 - self.y1*self.x2)*(other.x1-other.x2) - (self.x1-
self.x2)*(other.x1*other.y2 - other.y1*other.x2))/ \
         ((self.x1-self.x2)*(other.y1-other.y2) - (self.y1-self.y2)*(other.x1-
other.x2))
      y = ((self.x1*self.y2 - self.y1*self.x2)*(other.y1-other.y2) - (self.y1-
self.y2)*(other.x1*other.y2 - other.y1*other.x2))/ \
         ((self.x1-self.x2)*(other.y1-other.y2) - (self.y1-self.y2)*(other.x1-
other.x2))
      x = int(x)
      y = int(y)

      return x,y


import cv2
import numpy as np
```

Casilla

```
class Square:

    def __init__(self, image, c1, c2, c3, c4, position, state=''):

        # Esquinas
        self.c1 = c1
        self.c2 = c2
        self.c3 = c3
        self.c4 = c4

        # Posición
        self.position = position

        # npVector de esquinas
        self.contour = np.array([c1,c2,c4,c3],dtype=np.int32)

        # Centro del escaque
        M = cv2.moments(self.contour)
        cx = int(M['m10'] / M['m00'])
        cy = int(M['m01'] / M['m00'])

        # ROI para diferenciación de imágenes
        self.roi = (cx, cy)
        self.radius = 7

        self.emptyColor = self.roiColor(image)

        self.state = state

    def draw(self, image, color,thickness=2):

        # Formateo de npVectores de esquinas para drawContours

        ctr = np.array(self.contour).reshape((-1,1,2)).astype(np.int32)
        cv2.drawContours(image, [ctr], 0, color, 3)

    def drawROI(self, image, color, thickness = 1):

        cv2.circle(image, self.roi, self.radius, color,thickness)


    def roiColor(self, image):
        # Initializa máscara
        maskImage = np.zeros((image.shape[0], image.shape[1]), np.uint8)
        # Dibuja el círculo del ROI sobre la máscara
        cv2.circle(maskImage, self.roi, self.radius, (255, 255, 255), -1)
        # Encuentra el color promedio
        average_raw = cv2.mean(image, mask=maskImage)[::-1]
        # Necesita formato int, así que reasigna la variable
        average = (int(average_raw[1]), int(average_raw[2]),
int(average_raw[3]))

        return average

    def classify(self, image):

        rgb = self.roiColor(image)

        sum = 0
        for i in range(0,3):
            sum += (self.emptyColor[i] - rgb[i])**2

        cv2.putText(image, self.position,

self.roi,cv2.FONT_HERSHEY_SIMPLEX,0.5,(0,255,0),1,cv2.LINE_AA)
```

Reconocimiento de tablero

```
import math
import cv2
import numpy as np
from Line import Line
from Square import Square
from Board import Board

debug = True


class board_Recognition:
    '''
    Esta clase maneja la inicialización del tablero. Analiza el tablero vacío
encontrando su borde, líneas, esquinas,
    cuadrados, etc.
    '''

    def __init__(self, camera):

        self.cam = camera

    def initialize_Board(self):

        corners = []

        # Vuelve a tomar  la foto hasta que el tablero se inicialice
correctamente
        while len(corners) < 81:

            image = self.cam.get_frames()

            # Binariza la foto
            adaptiveThresh,img = self.clean_Image(image)

            # Apaga todos los píxeles fuera del borde del tablero de ajedrez
            mask = self.initialize_mask(adaptiveThresh,img)

            # Encuentra bordes
            edges,colorEdges = self.findEdges(mask)

            # Encuentra líneas
            horizontal, vertical = self.findLines(edges,colorEdges)

            # Encuentra esquinas
            corners = self.findCorners(horizontal, vertical, colorEdges)

        # Encuentra escaques
        squares = self.findSquares(corners, img)

        # Crea tablero
        board = Board(squares)

        return board

    def clean_Image(self,image):
        '''
        Convierte la foto a blanco y negro para un análisis más simple
        '''


        img = image

        # Convierte a escala de grises
        gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

        # Establece todos los píxeles por encima del valor de umbral en blanco y
los que están debajo en negro
        # Adaptive thresholding se utiliza para combatir las diferencias de
iluminación en la imagen.
        adaptiveThresh = cv2.adaptiveThreshold(gray, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 125, 1)
        if debug:
            # Muestra la imagen después de aplicado el umbral adaptativo
            cv2.imshow("Escala de grises", gray)
            cv2.imshow("Adaptive Thresholding", adaptiveThresh)
            cv2.waitKey(0)
            cv2.destroyAllWindows()

        return adaptiveThresh,img
```

```
   def initialize_mask(self, adaptiveThresh,img):
      '''
      Encuentra el borde del tablero de ajedrez y borra todos los píxeles
innecesarios
      '''

      # Encuentra contornos (polígonos cerrados)
      contours, hierarchy = cv2.findContours(adaptiveThresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

      # Crea copia de la imagen original
      imgContours = img.copy()

      for c in range(len(contours)):
         # Area
         area = cv2.contourArea(contours[c])
         # Perímetro
         perimeter = cv2.arcLength(contours[c], True)
            # Filtrar el borde del tablero de ajedrez, ya que algunos contornos
son tan pequeños que dan una división cero
            # Para valores de prueba son 70-40, para valores de tablero son 80-
75. Será necesario recalibrar si cambia
            # el cuadrado más grande es siempre la proporción más grande
         if c ==0:
            Lratio = 0
         if perimeter > 0:
            ratio = area / perimeter
            if ratio > Lratio:
               largest=contours[c]
               Lratio = ratio
               Lperimeter=perimeter
               Larea = area
         else:
               pass

      # Dibuja contornos
      cv2.drawContours(imgContours, [largest], -1, (0,0,0), 1)
      if debug:
         # Mostrar imagen con contornos dibujados
         cv2.imshow("Contornos tablero",imgContours)
         cv2.waitKey(0)
         cv2.destroyAllWindows()

      # El parámetro Epsilon es necesario para ajustar el contorno al polígono
      epsilon = 0.1 * Lperimeter
      # Aproxima un polígono del borde del tablero de ajedrez
      chessboardEdge = cv2.approxPolyDP(largest, epsilon, True)

      # Crea una nueva imagen totalmente negra
      mask = np.zeros((img.shape[0], img.shape[1]), 'uint8')*125
      # Copia los bordes del tablero de ajedrez como un polígono blanco del
tamaño del borde del tablero de ajedrez
      cv2.fillConvexPoly(mask, chessboardEdge, 255, 1)
      # Asigna todos los píxeles que son blancos (es decir, el polígono y el
tablero de ajedrez)
      extracted = np.zeros_like(img)
      extracted[mask == 255] = img[mask == 255]
      # retira la tira alrededor del borde
      extracted[np.where((extracted == [125, 125, 125]).all(axis=2))] = [0, 0,


20]

      if debug:
         # Muestra la imagen con la máscara dibujada
         cv2.imshow("Mascara",extracted)
         cv2.waitKey(0)
         cv2.destroyAllWindows()
      return extracted

   def findEdges(self, image):
      '''
      Encuentra bordes en la imagen. Más tarde los bordes serán utilizados para
encontrar líneas, etc.
      '''

      # Encuentra bordes
      edges = cv2.Canny(image, 100, 200, None, 3)
      if debug:
         # Muestra la imagen con los bordes dibujados
         cv2.imshow("Canny", edges)
         cv2.waitKey(0)
         cv2.destroyAllWindows()
```

```
        # Convierte la imagen de bordes a escala de grises
        colorEdges = cv2.cvtColor(edges,cv2.COLOR_GRAY2BGR)

        return edges,colorEdges

    def findLines (self, edges, colorEdges):
        '''
        Encuentra las líneas en la foto y las ordena en verticales y horizontales
        '''

        # Infiere líneas basadas en bordes
        lines = cv2.HoughLinesP(edges, 1,  np.pi / 180, 100,np.array([]), 100,
80)

        # Dibuja líneas
        a,b,c = lines.shape
        for i in range(a):
            cv2.line(colorEdges, (lines[i][0][0], lines[i][0][1]),
(lines[i][0][2], lines[i][0][3]), (0,255,0),2,cv2.LINE_AA)

        if  debug:
            # Muestra imagenes con líneas dibujadas
            cv2.imshow("Lines",colorEdges)
            cv2.waitKey(0)
            cv2.destroyAllWindows()

        # Crea objetos de línea y los ordena por orientación, vertical u
horizontal.
        horizontal = []
        vertical = []
        for l in range(a):
            [[x1,y1,x2,y2]] = lines[l]
            newLine = Line(x1,x2,y1,y2)
            if newLine.orientation == 'horizontal':
                horizontal.append(newLine)
            else:
                vertical.append(newLine)

        return horizontal, vertical

    def findCorners (self, horizontal, vertical, colorEdges):
        '''
        Encuentra esquinas en la intersección de líneas horizontales y verticales
        '''

        # Encuentra esquinas (intersecciones de líneas).
        corners = []
        for v in vertical:
            for h in horizontal:
                s1,s2 = v.find_intersection(h)
                corners.append([s1,s2])

        # Elimina esquinas duplicadas
        dedupeCorners = []
        for c in corners:
            matchingFlag = False
            for d in dedupeCorners:

                if math.sqrt((d[0]-c[0])*(d[0]-c[0]) + (d[1]-c[1])*(d[1]-c[1])) <
20:
                    matchingFlag = True
                    break
            if not matchingFlag:
                dedupeCorners.append(c)

        for d in dedupeCorners:
            cv2.circle(colorEdges, (d[0],d[1]), 10, (0,0,255))


        if debug:
            #Muestra imagen con esquinas encerradas en un círculo
            cv2.imshow("Esquinas",colorEdges)
            cv2.waitKey(0)
            cv2.destroyAllWindows()

        return dedupeCorners

    def findSquares(self, corners, colorEdges):
        '''
        Encuentra los cuadrados del tablero de ajedrez
        '''
```

```
        # Ordena esquinas por fila
        corners.sort(key=lambda x: x[0])
        rows = [[],[],[],[],[],[],[],[],[]]
        r = 0
        for c in range(0, 81):
            if c > 0 and c % 9 == 0:
                r = r + 1

            rows[r].append(corners[c])

        letters = ['a','b','c','d','e','f','g','h']
        numbers = ['1','2','3','4','5','6','7','8']
        Squares = []

        # Ordena esquinas por columna
        for r in rows:
            r.sort(key=lambda y: y[1])

        # Inicializa escaques
        for r in range(0,8):
            for c in range (0,8):
                c1 = rows[r][c]
                c2 = rows[r][c + 1]
                c3 = rows[r + 1][c]
                c4 = rows[r + 1][c + 1]

                position = letters[r] + numbers[7-c]
                newSquare = Square(colorEdges,c1,c2,c3,c4,position)
                newSquare.draw(colorEdges,(0,0,255),2)
                newSquare.drawROI(colorEdges,(255,0,0),2)
                newSquare.classify(colorEdges)
                Squares.append(newSquare)



        if debug:
            # Muestra imagen con cuadrados y la Región de Interes dibujados y la
posición etiquetada
            cv2.imshow("Escaques", colorEdges)
            cv2.waitKey(0)
            cv2.destroyAllWindows()

        return Squares
```

Tablero

```
import cv2
import math

debug = True

class Board:
   """
   Mantiene todas las instancias de la clase Square y actualiza los
cambios en el tablero después de los movimientos
   """
   def __init__(self, squares):

      self.squares = squares
      self.boardMatrix = []
      self.promotion = 'q'
      self.promo = False
      self.move = "e2e4"

   def draw(self,image):
      """
      Dibuja el tablero y clasifica los estado de los escaques en la
imagen
      """
      for square in self.squares:
         square.draw(image, (0,0,255))
         square.classify(image)

   def assignState(self):
      """
      Asigna estados iniciales de configuración a escaques e inicializa
la matriz del tablero
      """
      black = ['r', 'n', 'b','q','k','b','n','r']
      white = ['R','N','B','Q','K','B','N','R']

      for i in range(8):
         self.squares[8*i + 0].state = black[i]
         self.squares[8*i + 1].state = 'p'
         self.squares[8*i + 2].state = '.'
         self.squares[8*i + 3].state = '.'
         self.squares[8*i + 4].state = '.'
         self.squares[8*i + 5].state = '.'
         self.squares[8*i + 6].state = 'P'
         self.squares[8*i + 7].state = white[i]

      for square in self.squares:
         self.boardMatrix.append(square.state)

  def determineChanges(self,previous, current):
      '''
      Determina el cambio en los valores de color dentro de los
cuadrados de una imagen a otra para inferir el
      movimiento de la pieza
      '''

      copy = current.copy()

      largestSquare = 0
      secondLargestSquare = 0
      largestDist = 0
      secondLargestDist = 0
      stateChange = []

      # Comprueba las diferencias de color entre las fotos
      for sq in self.squares:
         colorPrevious = sq.roiColor(previous)
         colorCurrent = sq.roiColor(current)
```

```
            # Distancia en valores bgr
            sum = 0
            for i in range(0,3):
               sum += (colorCurrent[i] - colorPrevious[i])**2

            distance = math.sqrt(sum)

            if distance > 25:
               stateChange.append(sq)

            if distance > largestDist:
               # Actualiza escaques con el mayor cambio de color
               secondLargestSquare = largestSquare
               secondLargestDist = largestDist
               largestDist = distance
               largestSquare = sq

            elif distance > secondLargestDist:
               # Actualiza el segundo cambio de color
               secondLargestDist = distance
               secondLargestSquare = sq


         if  len(stateChange)  == 4:

            # Si cuatro escaques tienen cambio de color en un solo
movimiento, el enroque tuvo lugar
            squareOne = stateChange[0]
            squareTwo = stateChange[1]
            squareThree = stateChange[2]
            squareFour = stateChange[3]

            # Comprueba enroque corto en piezas blancas
            if squareOne.position == "e1" or squareTwo.position == "e1" or
squareThree.position == "e1" or  squareFour.position == "e1":
               if squareOne.position == "f1"  or squareTwo.position == "f1"
or squareThree.position == "f1"  or squareFour.position == "f1":
                  if squareOne.position == "g1" or squareTwo.position ==
"g1" or squareThree.position == "g1" or  squareFour.position == "g1":
                     if squareOne.position == "h1"  or squareTwo.position
== "h1" or squareThree.position == "h1"  or squareFour.position == "h1":
                        self.move = "e1g1"
                        print(self.move)
                        if debug:
                           squareOne.draw(copy, (255,0,0), 2)
                           squareTwo.draw(copy, (255,0,0), 2)
                           squareThree.draw(copy, (255,0,0),2)
                           squareFour.draw(copy, (255,0,0), 2)
                           cv2.imshow("previous",previous)
                           cv2.imshow("identified",copy)
                           cv2.waitKey()
                           cv2.destroyAllWindows()
                        return self.move

               # Comprueba enroque largo en piezas blancas
               if squareOne.position == "d1"  or squareTwo.position == "d1"
or squareThree.position == "d1"  or squareFour.position == "d1":
                  if squareOne.position == "c1"  or squareTwo.position ==
"c1" or squareThree.position == "c1"  or squareFour.position == "c1":
                     if squareOne.position == "a1"  or squareTwo.position
== "a1" or squareThree.position == "a1"  or squareFour.position == "a1":

                        self.move = "e1c1"
                        print(self.move)
                        if debug:
```

```
                        squareOne.draw(copy, (255,0,0), 2)
                        squareTwo.draw(copy, (255,0,0), 2)
                        squareThree.draw(copy, (255,0,0),2)
                        squareFour.draw(copy, (255,0,0), 2)
                        cv2.imshow("previous",previous)
                        cv2.imshow("identified",copy)
                        cv2.waitKey()
                        cv2.destroyAllWindows()
                    return self.move

        # Comprueba enroque corto en piezas negras
        if squareOne.position == "e8" or squareTwo.position == "e8" or
squareThree.position == "e8" or  squareFour.position == "e8":
            if squareOne.position == "f8"  or squareTwo.position == "f8"
or squareThree.position == "f8"  or squareFour.position == "f8":
                if squareOne.position == "g8"  or squareTwo.position ==
"g8" or squareThree.position == "g8"  or squareFour.position == "g8":
                    if squareOne.position == "h8"  or squareTwo.position
== "h8" or squareThree.position == "h8"  or squareFour.position == "h8":
                        self.move = "e8g8"
                        print(self.move)
                        if debug:
                            squareOne.draw(copy, (255,0,0), 2)
                            squareTwo.draw(copy, (255,0,0), 2)
                            squareThree.draw(copy, (255,0,0),2)
                            squareFour.draw(copy, (255,0,0), 2)
                            cv2.imshow("previous",previous)
                            cv2.imshow("identified",copy)
                            cv2.waitKey()
                            cv2.destroyAllWindows()
                        return self.move


            # Comprueba enroque largo en piezas blancas
            if squareOne.position == "d8"  or squareTwo.position == "d8"
or squareThree.position == "d8"  or squareFour.position == "d8":
                if squareOne.position == "c8"  or squareTwo.position ==
"c8" or squareThree.position == "c8"  or squareFour.position == "c8":
                    if squareOne.position == "a8"  or squareTwo.position
== "a8" or squareThree.position == "a8"  or squareFour.position == "a8":
                        self.move = "e8c8"
                        print(self.move)
                        if debug:
                            squareOne.draw(copy, (255,0,0), 2)
                            squareTwo.draw(copy, (255,0,0), 2)
                            squareThree.draw(copy, (255,0,0),2)
                            squareFour.draw(copy, (255,0,0), 2)
                            cv2.imshow("previous",previous)
                            cv2.imshow("identified",copy)
                            cv2.waitKey()
                            cv2.destroyAllWindows()
                        return self.move


    # Movimiento regular, dos escaques cambian de estado
    squareOne = largestSquare
    squareTwo = secondLargestSquare

    if debug:
        squareOne.draw(copy, (255,0,0), 2)
        squareTwo.draw(copy, (255,0,0), 2)
        cv2.imshow("previo",previous)
        cv2.imshow("identificado",copy)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    # Obtiene colores para cada escaque de cada foto
    oneCurr = squareOne.roiColor(current)
    twoCurr = squareTwo.roiColor(current)
```

```
      # Calcula la distancia del valor de color del escaque vacío
      sumCurr1 = 0
      sumCurr2 = 0
      for i in range(0,3):
         sumCurr1 += (oneCurr[i] - squareOne.emptyColor[i])**2
         sumCurr2 += (twoCurr[i] - squareTwo.emptyColor[i])**2

      distCurr1 = math.sqrt(sumCurr1)
      distCurr2 = math.sqrt(sumCurr2)

      if distCurr1 < distCurr2:
         # Si el escaque 1 está más cerca del valor de color vacío,
entonces está vacío.
         squareTwo.state = squareOne.state
         squareOne.state = '.'
         # Revisa la coronación de un peón
         if squareTwo.state.lower() == 'p':
            if squareOne.position[1:2] == '2' and
squareTwo.position[1:2] == '1':
               self.promo = True
            if squareOne.position[1:2] == '7' and
squareTwo.position[1:2] == '8':
               self.promo = True

         self.move = squareOne.position + squareTwo.position

      else:
         # El escaque 2 está actualmente vacío
         squareOne.state = squareTwo.state
         squareTwo.state = '.'
         # Revisa la coronación de un peón
         if squareOne.state.lower() == 'p':
            if squareOne.position[1:2] == '1' and
squareTwo.position[1:2] == '2':
               self.promo = True
            if squareOne.position[1:2] == '8' and
squareTwo.position[1:2] == '7':
               self.promo = True


         self.move = squareTwo.position + squareOne.position

      return self.move
```

Motor de ajedrez

```
import chess
import chess.engine
import socket

#Create a socket object
s = socket.socket()
port = 55000
s.connect(('127.0.0.1', port))

class ChessEng:
    '''
    Esta clase interactúa con el motor de ajedrez stockfish usando el paquete
python-chess.
    Todas las interacciones se realizan con el Protocolo de Interfaz Universal
de Ajedrez (UCI).
    Trasncribe el juego a un archivo txt llamado Game.txt
    '''

    def __init__(self):
        '''
        Crea el tablero de ajedrez, el motor de ajedrez e inicia el protocolo
UCI.
        '''

        self.engBoard = chess.Board()
        self.engine =
chess.engine.SimpleEngine.popen_uci("./stockfish/Windows/stockfish_10_x64")
        print(self.engBoard)



    def updateMove(self, move):
        '''
        Actualiza el tablero de ajedrez con el movimiento realizado.
        También verifica movimientos ilegales
        '''

        # Convierte el movimiento a formato UCI para el motor de ajedrez
        uciMove = chess.Move.from_uci(move)

        # Comprueba la legalidad
        if uciMove not in self.engBoard.legal_moves:
            return 1
        else:
            # Actualiza el tablero
            self.engBoard.push(uciMove)
            print(self.engBoard)
            return 0


    def feedToAI(self):
        '''
        Obtiene el mejor movimiento del motor stockfish. Escribe el movimiento
 elegido al archivo Game.txt
        '''
      # Da al CPU la posición actual del tablero
      response = self.engine.play(self.engBoard,
chess.engine.Limit(time=0.200))

      # Da información sobre qué tipo de movimiento es, ataque o enroque.

      attack = str(self.engBoard.is_capture(response.move))
      castling = str(self.engBoard.is_castling(response.move))

      movetoRobot = response.move.uci()

      a,b = movetoRobot[:2], movetoRobot[2:]

      # Actualiza el tablero
      self.engBoard.push(response.move)


      i = 0

      while True:
          if i == 0:
              s.send(bytes(a, 'utf-8'))
              i = i + 1
              s.recv(4096)
          elif i == 1:
```

```
            s.recv(4096)
        elif i == 1:
            s.send(bytes(b, 'utf-8'))
            i = i + 1
            s.recv(4096)
        elif i == 2:
            s.send(bytes(attack, 'utf-8'))
            i = i + 1
            s.recv(4096)
        elif i == 3:
            s.send(bytes(castling, 'utf-8'))
            i = i + 1
            s.recv(4096)
        elif i > 3:
            break

    # Escribe mover al archivo txt
    f = open("Game.txt", "a+")
    f.write(response.move.uci()+ "\r\n")
    f.close()

    print(attack)
    print(response.move.uci())
    print(self.engBoard)

    return response.move
```

Juego

```
import chess
from Camera import Camera
from ChessEng import ChessEng
from board_Recognition import board_Recognition
from python_socket import Socket

class Game:
   '''
   Esta clase contiene información del juego que interactúa con el tablero y el
motor de ajedrez
   '''

   def __init__(self):
      '''
      Inicializa el objeto Game y crea varios valores booleanos con respecto al
estado del juego.
      Establece el ganador del juego como marcador de posición
      '''
      self.over = False
      self.CPUMoveError = False
      self.PlayerMoveError = False
      self.isCheck = False
      self.winner = "YO"

   def setUp(self):

      self.camera = Camera()
      self.chessEngine = ChessEng()
      self.zocalo = Socket()
      self.board = 0
      self.current = 0
      self.previous = 0
      self.CPULastMove = "0"


   def analyzeBoard(self):
      '''
      Llama a board_recognition para tomar una imagen e inicializa Board
      '''

      boardRec = board_Recognition(self.camera)
      self.board = boardRec.initialize_Board()
      self.board.assignState()

   def checkBoardIsSet(self):

      self.current = self.camera.get_frames()


   def playerMove(self):
      '''
      Compara el tablero anterior con el tablero actual para determinar el
movimiento realizado
      por el jugador
      '''
      self.previous = self.current
      self.current = self.camera.get_frames()
      move = self.board.determineChanges(self.previous,self.current)
      code = self.chessEngine.updateMove(move)
      if code == 1:
         # Movimiento ilegal despliega una página de error de movimiento del
jugador
         self.PlayerMoveError = True
      else:
         self.PlayerMoveError = False
         # Escribe al archivo Game.txt
         f = open("Game.txt", "a+")       # "a+", para escribir sobre un
fichero existente añadiendo al final
         f.write(chess.Move.from_uci(move).uci() + "\r\n")         # "\r\n",
para añadir nueva línea
         f.close()

      # Comprueba si se acabo el juego
      if  self.chessEngine.engBoard.is_checkmate():
         self.winner = "¡Tú ganas!"
         self.over = True

   def playerPromotion(self, move):
      '''
      Compara el tablero anterior con el tablero actual para determinar el
movimiento realizado por el jugador
      '''
```

```
            print(move)
            code = self.chessEngine.updateMove(move)
            if code == 1:
                # Movimiento ilegal despliega una página de error de movimiento del
jugador
                print("Error")
                self.PlayerMoveError = True
            else:
                self.PlayerMoveError = False

                # Escribe al archivo Game.txt
                f = open("Game.txt", "a+")
                f.write(chess.Move.from_uci(move).uci() + "\r\n")
                f.close()

            # Comprueba si se acabo el juego
            if  self.chessEngine.engBoard.is_checkmate():
                self.winner = "¡Tú ganas!"
                self.over = True


        def CPUMove(self):

            # Obtiene el movimiento desde el motor de ajedrez
            self.CPULastMove = self.chessEngine.feedToAI()

            # Si está en jaque el GUI abrirá una ventana alertando al usuario del
Jaque
            self.isCheck = self.chessEngine.engBoard.is_check()

            # Comprueba si se acabo el juego
            if self.chessEngine.engBoard.is_checkmate():
                self.winner = "¡CPU gana!"
                self.over = True

            return self.CPULastMove

        def updateCurrent(self):

            self.previous = self.current
            self.current = self.camera.get_frames()

            # determina el movimiento
            move = self.board.determineChanges(self.previous, self.current)
            move = chess.Move.from_uci(move)

            # Asegura que el jugador haya movido la pieza de la CPU correctamente
            if move == self.CPULastMove:
                self.CPUMoveError = False
            else:
                # GUI abrirá una página de error de movimiento del CPU
                self.CPUMoveError = True
```

Interfaz gráfica de usuario

```
import tkinter as tk
from tkinter import *
from Game import Game


# Establece tamaños de fuente
LARGE_FONT = ("system", 20)
MED_FONT = ("system", 12)
SMALL_FONT = ("system", 8)


class Application(tk.Tk):
   '''
   Esta clase controla la Interfaz Gráfica de Usuario (GUI)
   '''
   def __init__(self,*args, **kwargs):

      tk.Tk.__init__(self,*args,**kwargs)

      container = tk.Frame(self)

      container.pack(side = "top", fill = "both", expand = True)

      container.grid_rowconfigure(0,weight = 1)
      container.grid_columnconfigure(0, weight = 1)

      self.frames = {}
      self.game = Game()

      # Contiene la información de movimiento de la CPU que se mostrará en
CPUMovePage
      self.move = StringVar()
      self.move.set("e2")

      # Contiene la información del ganador que se mostrará en GameOverPage
      self.winner = StringVar()
      self.winner.set("¡CPU gana!")

      # Da objetos de página a la aplicación para mostrar en el frame
      for F in (StartGamePage, InitializeBoardPage,SetBoardPage,
ChooseColorPage,
            ChooseDifficultyPage, CPUMovePage, PlayerMovePage, CheckPage,
            CPUMoveErrorPage, GameOverPage, PlayerMoveErrorPage,
ChoosePromotionPage):

         frame = F(container, self)
         self.frames[F] = frame
         frame.grid(row = 0, column = 0, sticky = "nsew")

      self.show_frame(StartGamePage)

   def show_frame(self,cont):
      '''
      Eleva el marco a la parte superior, mostrándolo como la ventana actual
      '''

      frame = self.frames[cont]
      frame.tkraise()

class StartGamePage(tk.Frame):
   '''
   Solicita al usuario que comience un juego nuevo
   '''

   def __init__(self,parent,controller):

      tk.Frame.__init__(self,parent)
      controller.title("Robot autónomo para juego de ajedrez")
      controller.iconbitmap('logo_UPC.ico')
      controller.resizable(width=False, height=False)
      # Coloca una etiqueta
      label = tk.Label(self,text = "Robot Ajedrecista, usando un sistema de
visión artificial", font = LARGE_FONT)
      label.pack(pady = 20, padx = 20)

      # Crea el botón que te lleva a la página de inicialización del tablero y
llama a Game.setUp()
      startGameButton = tk.Button(self, text = "Comenzar Juego Nuevo",font =
MED_FONT,
```

```
                command = lambda:
[controller.show_frame(InitializeBoardPage), controller.game.setUp()])
      startGameButton.pack()


class InitializeBoardPage(tk.Frame):
   '''
   Pide al jugador que despeje el tablero, para que este pueda inicializarse
   '''

   def __init__(self,parent,controller):

      tk.Frame.__init__(self,parent)
      label = tk.Label(self,text = "Despeja el tablero para la configuración
del juego", font = LARGE_FONT)
      label.pack(pady = 10, padx = 10)
      initBoardButton = tk.Button(self, text = "Hecho",font = MED_FONT, command
= lambda : [controller.show_frame(SetBoardPage),
controller.game.analyzeBoard()])
      initBoardButton.pack()

class ChooseColorPage(tk.Frame):
   '''
   Pide al jugador que elija el color y muestra la ventana apropiada para el
primer movimiento.
   '''

   def __init__(self,parent,controller):

      tk.Frame.__init__(self,parent)

      # Coloca una etiqueta
      label = tk.Label(self,text = "¿De qué color te gustaría jugar?", font =
LARGE_FONT)
      label.pack(pady = 10, padx = 10)

      # Coloca el botón que te llevará a la Página de Movimiento del CPU ya que
irá primero
      # gets move from CPU and displays it in the next window
      blackButton = tk.Button(self, text = "Rojas (Negras)",font = MED_FONT,
               command = lambda :
[controller.show_frame(CPUMovePage),controller.move.set(controller.game.CPUMove
())])
      blackButton.pack()

      # Coloca el botón que te llevará a la Página de Movimiento del jugador ya
que irá primero
      whiteButton = tk.Button(self, text = "Verdes (Blancas)",font = MED_FONT,
               command = lambda: controller.show_frame(PlayerMovePage))
      whiteButton.pack()

class SetBoardPage(tk.Frame):
   '''
   Solicita al usuario que configure el tablero después de la inicialización
   '''

   def __init__(self,parent,controller):

      tk.Frame.__init__(self,parent)
      # Coloca una etiqueta
      label = tk.Label(self,text = "Inicialización del juego realizada. Pon la
piezas", font = LARGE_FONT)
      label.pack(pady = 10, padx = 10)

      # Coloca el botón que te lleva a elegir página de dificultad y hace que
el Juego tome una foto del Tablero configurado
      setBoardButton = tk.Button(self, text = "Hecho",font = MED_FONT,
                  command = lambda :
[controller.show_frame(ChooseDifficultyPage),controller.game.checkBoardIsSet()]
)
      setBoardButton.pack()

class PlayerMovePage(tk.Frame):
   '''
   Solicita al jugador que realice su movimiento
   '''

   def __init__(self,parent,controller):

      tk.Frame.__init__(self,parent)

      # Coloca una etiqueta
      label = tk.Label(self,text = "Tu movimiento", font = LARGE_FONT)
      label.pack(pady = 10, padx = 10)
```

```
      # Coloca el botón que actualiza el movimiento del jugador y verifica la
validez de los movimientos y las circunstancias del tablero
      PlayerButton = tk.Button(self, text = "Hecho",font = MED_FONT,
                  command = lambda :
[controller.game.playerMove(),self.checkValid(controller)])

      # Coloca el botón que termina el juego. Muestra el juego terminado sobre
la página con el CPU como ganador
      ResignButton = tk.Button(self, text = "Renunciar",font = SMALL_FONT,
                  command = lambda : controller.show_frame(GameOverPage))
      PlayerButton.pack()
      ResignButton.pack()

   def checkValid(self,controller):
      '''
      Realiza varios controles sobre la validez del movimiento y las
circunstancias del tablero.
      Muestra la ventana apropiada dadas las condiciones.
      '''

      if controller.game.over:
         controller.winner.set(controller.game.winner)
         controller.show_frame(GameOverPage)

      elif controller.game.board.promo:
         controller.show_frame(ChoosePromotionPage)

      elif controller.game.PlayerMoveError:
         controller.game.current = controller.game.previous
         controller.show_frame(PlayerMoveErrorPage)

      else:
         controller.move.set(controller.game.CPUMove())
         controller.show_frame(CPUMovePage)

class CPUMovePage(tk.Frame):
   '''
   Muestra el movimiento del motor de ajedrez y solicita al usuario que mueva
la pieza
   '''

   def __init__(self,parent,controller):

      tk.Frame.__init__(self,parent)


      # Coloca una etiqueta
      label = tk.Label(self,text = "Movimiento del CPU:", font = LARGE_FONT)
      label.pack(pady = 10, padx = 10)


      # Coloca una etiqueta dinámica con el movimiento del CPU
      self.moveLabel = tk.Label(self, textvariable=controller.move,
font=MED_FONT)
      self.moveLabel.pack(pady=10, padx=10)



      # Coloca el botón que actualiza las fotos del tablero después de que el
 usuario mueve la pieza del CPU. Verifica la validez del movimiento
       CPUButton = tk.Button(self, text = "Hecho",font = MED_FONT,
                   command = lambda : [controller.game.updateCurrent(),
 self.checkValid(controller)])
       CPUButton.pack()

    def checkValid(self,controller):
       '''
       Realiza varios controles sobre la validez del movimiento y las
 circunstancias del tablero.
       Muestra la ventana apropiada dadas las condiciones.
       '''
       if controller.game.over:
          controller.winner.set(controller.game.winner)
          controller.show_frame(GameOverPage)

       elif controller.game.isCheck:
          controller.show_frame(CheckPage)

       elif controller.game.CPUMoveError:
          controller.game.current = controller.game.previous
          controller.show_frame(CPUMoveErrorPage)
```

```
        else:
            controller.show_frame(PlayerMovePage)

class CheckPage(tk.Frame):
    '''
    Alerta al usuario que está en Jaque
    '''

    def __init__(self,parent,controller):

        tk.Frame.__init__(self,parent)

        # Coloca una etiqueta
        label = tk.Label(self,text = "Estás en Jaque", font = LARGE_FONT)
        label.pack(pady = 10, padx = 10)

        # Coloca el botón que muestra la página de movimiento del jugador
        setBoardButton = tk.Button(self, text = "Proceder",font = MED_FONT,
                            command = lambda :
controller.show_frame(PlayerMovePage))
        setBoardButton.pack()

class CPUMoveErrorPage(tk.Frame):
    '''
    Alerta al usuario de que el movimiento que realizó no es el mismo que la CPU
solicitó
    '''

    def __init__(self,parent,controller):

        tk.Frame.__init__(self,parent)

        # Coloca etiqueta
        label = tk.Label(self,text = "Ese no fue el movimiento correcto de la
CPU", font = LARGE_FONT)
        label.pack(pady = 10, padx = 10)

        # Coloca el botón que muestra CPUMovePage
        setBoardButton = tk.Button(self, text = "Inténtalo de nuevo",font =
MED_FONT,
                    command = lambda : controller.show_frame(CPUMovePage))
        setBoardButton.pack()


class PlayerMoveErrorPage(tk.Frame):
    '''
    Alerta al usuario que hizo un movimiento no válido
    '''

    def __init__(self,parent,controller):

        tk.Frame.__init__(self,parent)

        # Coloca etiqueta
        label = tk.Label(self,text = "Error Movimiento No Válido", font =
LARGE_FONT)
        label.pack(pady = 10, padx = 10)

        # Coloca el botón que muestra la PlayerMovePage

        setBoardButton = tk.Button(self, text = "Inténtalo de nuevo", font =
MED_FONT,
                    command = lambda : controller.show_frame(PlayerMovePage))
        setBoardButton.pack()

class GameOverPage(tk.Frame):
    '''
    Muestra al ganador del juego
    '''

    def __init__(self,parent,controller):

        tk.Frame.__init__(self,parent)

        # Coloca etiqueta
        label = tk.Label(self,text = "Game Over", font = LARGE_FONT)

        # Establece etiqueta dinámica que se actualizará con el ganador del juego
        self.winnerLabel = tk.Label(self, textvariable = controller.winner, font
= LARGE_FONT)
        label.pack(pady = 10, padx = 10)
        self.winnerLabel.pack(pady = 10, padx = 10)
```

```
class ChooseDifficultyPage(tk.Frame):
   '''
   Pide al usuario que elija una dificultad para el motor de ajedrez
   '''

   def __init__(self,parent,controller):

      tk.Frame.__init__(self,parent)
      label = tk.Label(self,text = "Elije la dificultad")
      label.pack(pady = 10, padx = 10)

      EasyButton = tk.Button(self, text = "Facil",
                  command = lambda : [self.setEasy(controller),
controller.show_frame(ChooseColorPage)])
      EasyButton.pack()

      IntermediateButton = tk.Button(self, text = "Intermedio",
                  command = lambda : [self.setIntermediate(controller),
controller.show_frame(ChooseColorPage)])
      IntermediateButton.pack()

      HardButton = tk.Button(self, text = "Duro",
                  command = lambda : [self.setHard(controller),
controller.show_frame(ChooseColorPage)])
      HardButton.pack()


      ExtremeButton = tk.Button(self, text = "Extremo",
                  command = lambda : [self.setExtreme(controller),
controller.show_frame(ChooseColorPage)])

      ExtremeButton.pack()

      MasterButton = tk.Button(self, text = "Maestro",
                  command = lambda : [self.setMaster(controller),
controller.show_frame(ChooseColorPage)])

      MasterButton.pack()
   def setEasy(self,controller):
      controller.game.chessEngine.engine.configure({'Skill Level' : 1})

   def setIntermediate(self,controller):
      controller.game.chessEngine.engine.configure({'Skill Level' : 5})

   def setHard(self,controller):
      controller.game.chessEngine.engine.configure({'Skill Level' : 10})

   def setExtreme(self,controller):
      controller.game.chessEngine.engine.configure({'Skill Level' : 15})

   def setMaster(self,controller):
      controller.game.chessEngine.engine.configure({'Skill Level' : 20})

class ChoosePromotionPage(tk.Frame):
   '''
   Solicita al usuario que elija a qué pieza le gustaría promocionar a su peón
   '''

   def __init__(self,parent,controller):

       tk.Frame.__init__(self,parent)
       label = tk.Label(self,text = "Elige tu promoción")
       label.pack(pady = 10, padx = 10)

       QueenButton = tk.Button(self, text = "Reina",
                   command = lambda : [self.setQueen(controller)])

       RookButton = tk.Button(self, text = "Torre",
                   command = lambda : [self.setRook(controller)])

       BishopButton = tk.Button(self, text = "Alfil",
                   command = lambda : [self.setBishop(controller)])

       KnightButton = tk.Button(self, text = "Caballo",
                   command = lambda : [self.setKnight(controller)])
       QueenButton.pack()
       RookButton.pack()
       BishopButton.pack()
       KnightButton.pack()
```

```
   def setQueen(self,controller):
      '''
      Actualiza el movimiento a un movimiento reconocido por la UCI para la
promoción.
      Revisa la validez y actualiza el tablero.
      '''

      controller.game.board.promotion = 'q'
      controller.game.board.move = controller.game.board.move + 'q'
      controller.game.playerPromotion(controller.game.board.move)

      if controller.game.PlayerMoveError:
         controller.game.current = controller.game.previous
         controller.show_frame(PlayerMoveErrorPage)
      else:
         controller.move.set(controller.game.CPUMove())
         controller.show_frame(CPUMovePage)

   def setRook(self,controller):
      '''
      Actualiza el movimiento a un movimiento reconocido por la UCI para la
promoción.
      Revisa la validez y actualiza el tablero.
      '''

      controller.game.board.promotion = 'r'
      controller.game.board.move = controller.game.board.move + 'r'
      controller.game.playerPromotion(controller.game.board.move)

      if controller.game.PlayerMoveError:
         controller.game.current = controller.game.previous
         controller.show_frame(PlayerMoveErrorPage)
      else:
         controller.move.set(controller.game.CPUMove())
         controller.show_frame(CPUMovePage)

   def setBishop(self,controller):
      '''
      Actualiza el movimiento a un movimiento reconocido por la UCI para la
promoción.
      Revisa la validez y actualiza el tablero.
      '''

      controller.game.board.promotion = 'b'
      controller.game.board.move = controller.game.board.move + 'b'
      controller.game.playerPromotion(controller.game.board.move)

      if controller.game.PlayerMoveError:
         controller.game.current = controller.game.previous
         controller.show_frame(PlayerMoveErrorPage)
      else:
         controller.move.set(controller.game.CPUMove())
         controller.show_frame(CPUMovePage)

   def setKnight(self,controller):
      '''
      Actualiza el movimiento a un movimiento reconocido por la UCI para la
promoción.
      Revisa la validez y actualiza el tablero.
       '''
       controller.game.board.promotion = 'n'
       controller.game.board.move = controller.game.board.move + 'n'
       controller.game.playerPromotion(controller.game.board.move)

       if controller.game.PlayerMoveError:
          controller.game.current = controller.game.previous
          controller.show_frame(PlayerMoveErrorPage)
       else:
          controller.move.set(controller.game.CPUMove())
          controller.show_frame(CPUMovePage)


app = Application()
app.mainloop()
```